20th International Workshop on Project Management and Scheduling

13-16 Apr 2026
TOULOUSE
France

Table of contents

14-14 Apr 2026

A - Project management

A generalization of Hu's algorithm for resource leveling

Péter Györgyi[1] and Tamás Kis[1]

HUN-REN SZTAKI
gyorgyi.peter, kis.tamas@sztaki.hu

Keywords: scheduling, resource leveling, precedence constraints.

1 Introduction

Resource leveling is a traditional challenge in project scheduling that focuses on managing the demand for shared resources across multiple projects. A primary goal is to maintain consistent workload levels, which is the central focus of this paper. Due to the problem's complexity, polynomial-time optimization or approximation algorithms are generally infeasible, leading to the predominance of exact methods and heuristics in this area (Ballestín *et. al.* (2007), Hartmann and Birskorn (2022)). The problem has several practical applications, e.g., in construction industry (Kreter *et al.* 2014) and in chemical engineering (Megow *et al.* 2011).

In this paper we study a variant of the resource leveling problem, in which there is a set of n unit-time jobs, each one requiring one unit from a common resource. The jobs have a common deadline d. There is a precedence relation among the jobs inducing a set of in-trees. In addition, there is a convex function f of the resource usage. A schedule S specifies a start time $S_j \in \{0, \dots, d-1\}$ for each job j. A schedule is feasible if it respects the precedence constraints, that is, $S_i < S_j$ whenever job i precedes job j in the precedence relation. The cost of a schedule S is $cost(S) := \sum_{t=0}^{d-1} f(\ell^S(t))$, where $\ell^S(t)$ is the resource usage in time interval $[t, t+1]$. The goal is to find a feasible schedule of minimum cost. Note that a feasible schedules exists if and only if the length of the longest directed path in the precedence graph is at most d. We emphasize that f can be any convex function, which permits to model some popular objective functions in resource leveling. For instance, using $f(\ell) = \max\{0, \ell - c\}$, we can penalize resource usage exceeding a threshold c, which was studied by (Bendotti 2024). If $f(\ell) = (\ell - c)^2$, then the squared deviation from c is minimized.

2 Preliminaries

The *height* of job j, denoted by $h(j)$ is the length of the path between j and the root of its in-tree in the precedence graph. Let r be such that $h(j) \leq r$ for each job j. Let $V_i := \{j : h(j) = i\}$ for $i = 0, \dots, r$. The *scheduling graph* G has a set of nodes $N = \{0\} \cup \{d-r, \dots, d\}$ and a set of edges $E = \{(a, b) \in N \times N \mid a < b\}$. The set of jobs $\mathcal{J}_{(t,u)}$ corresponding to an edge (t, u) of G is $\bigcup_{i=x}^{y} V_i$, where $x = d - u$, and $y = r$ if $t = 0$, and $y = d - t - 1$ otherwise. Let $\alpha_{(t,u)} = |\mathcal{J}_{(t,u)}|/(u-t)$. For any edge (t, u) of G, its *ideal profile* is a mapping $p : \{t, \dots, u-1\} \longrightarrow \mathbb{Z}_{>0}$ such that $\sum_{\tau=t}^{u-1} p(\tau) = |\mathcal{J}_{(t,u)}|$ and $\lceil \alpha_{(t,u)} \rceil = p(t) \geq p(t+1) \geq \cdots \geq p(u-1) = \lfloor \alpha_{(t,u)} \rfloor$. Observe that for any edge (t, u) of G, the ideal profile p always exists and is unique.

We construct the optimal schedule by concatenating partial schedules corresponding to the edges of a path P of the scheduling graph G from node 0 to node d. For an edge $(t, u) \in P$, the jobs of $\mathcal{J}_{(t,u)}$ will be scheduled in the time interval $[t, u]$. The shape of such a partial schedule will be determined by the ideal profile of (t, u). However, such a schedule may not be feasible for an edge (t, u).

Definition 1. *Let (t,u) be an edge of G, and p its ideal profile. A schedule S of $\mathcal{J}_{(t,u)}$ specifies a time point $S(j) \in \{t,\ldots,u-1\}$ for each $j \in \mathcal{J}_{(t,u)}$, and it is* feasible *for p if $|\{j \in \mathcal{J}_{(t,u)} \mid S(j)=\tau\}| = p(\tau)$ for each $\tau \in \{t,\ldots,u-1\}$, while $S(j) < S(j')$ for each pair of jobs such that j must precede j'. The ideal profile p is* feasible *if there exists a feasible schedule for p.*

Note that if the ideal profile of (t,u) is feasible, then the cost of the feasible schedule of $\mathcal{J}_{(t,u)}$ in $[t,u]$ is $\sum_{\tau=t}^{u-1} f(p_\tau)$, and one can prove that any other schedule of $\mathcal{J}_{(t,u)}$ in $[t,u]$ cannot have a smaller cost.

Definition 2. *Let (t,u) be an edge of G, and p its ideal profile. Consider an assignment (mapping) $\sigma : \mathcal{J}_{(t,u)} \longrightarrow \{t,\ldots,u-1\}$. We say that σ satisfies the* Generalized Hu Condition (shortly GHC) *for p if (i) $|\{j \in \mathcal{J}_{(t,u)} : \sigma(j)=\tau\}| = p(\tau)$ for each time point $\tau \in \{t,\ldots,u-1\}$, and (ii) $\sigma(j)+h(j) \le d-1$ for each job $j \in \mathcal{J}_{(t,u)}$. We say that an ideal profile p is* GHC *if there exists an assignment σ as above satisfying the GHC for p.*

Note that the assignment σ in the above definition may not respect the precedence constraints among the jobs of $\mathcal{J}_{(t,u)}$.

Example 1. Consider an instance with jobs $j_1, j_2, \ldots, j_{22}$, and deadline $d = 8$. Figure 1(a) depicts the precedence graph. The red ellipses indicate the jobs with the same height, e.g., jobs with height $h(j) = 3$ are in V_3. The corresponding scheduling graph G is depicted in Figure 1(b). The blue rectangle in Figure 1(a) indicates the jobs correspond to edge $(3,7)$ of G, i.e., the jobs of $\mathcal{J}_{(3,7)}$. Since $|\mathcal{J}_{(3,7)}| = 9$, $p = (3,2,2,2)$ is the ideal profile of edge $(3,7)$. Let $\sigma(j_2) = \sigma(j_3) = 6$, $\sigma(j_4) = \sigma(j_5) = 5$, $\sigma(j_6) = \sigma(j_7) = 4$, $\sigma(j_8) = \sigma(j_9) = \sigma(j_{10}) = 3$ be an assignment of jobs of $\mathcal{J}_{(3,7)}$ to time points in $\{3,\ldots,6\}$. Note that σ is not a feasible schedule for these jobs, since $\sigma(j_5) = \sigma(j_4)$. Nevertheless, σ satisfies the GHC for ideal profile p, thus p is GHC.

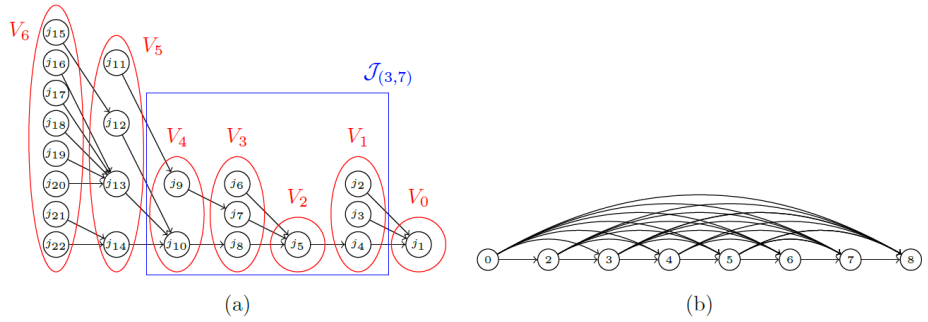


Fig. 1. (a) precedence graph for the example, (b) the corresponding scheduling graph G.

Consider any edge (t,u) of G, and its ideal profile p. If p is feasible, then the feasible schedule of $\mathcal{J}_{(t,u)}$ in the interval $[t,u]$ constitutes an assignment of $\mathcal{J}_{(t,u)}$ to time points in $\{t,\ldots,u-1\}$ which satisfies the GHC. Hence, p is GHC. One can prove that the converse also holds using a modified version of the method of (Hu 1961).

Theorem 1. *The ideal profile p for an edge (t,u) of G is GHC if and only if p is feasible.*

We can further filter the edges of G by the following definitions.

Definition 3. *We say that edge (t,u) of G is* good *if its ideal profile is feasible. Edge (t,u) is a* maximal good edge *if it is good, and there exists no $t_1 < t$ such that (t_1,u) is a good edge of G. We call a feasible schedule for the ideal profile of the edge an* ideal schedule.

As it will turn out in the next section, there exists an optimal solution which corresponds to a path P in G from node 0 to node d consisting of maximal good edges only.

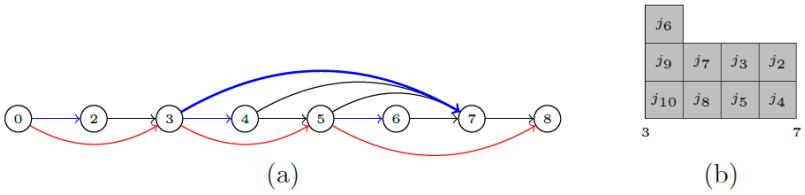


Fig. 2. (a) Good edges of G, and (b) ideal schedule for edge $(3,7)$.

Observe that a good edge always has an ideal schedule and edge (t,u) of G is good if and only if its ideal profile is GHC. To decide if an edge (t,u) of G is good, we define an assignment $\sigma_{(t,u)}$ as follows. We index the jobs of $\mathcal{J}_{(t,u)}$ in monotone decreasing height order, that is, if j, j' are distinct jobs in $\mathcal{J}_{(t,u)}$, then $j \prec j'$ if $h(j) \geq h(j')$. Consider the ideal profile p of edge (t,u). We assign the first $p(t)$ jobs of $\mathcal{J}_{(t,u)}$ (in the $\prec$ order) to time point t, the next $p(t+1)$ jobs to time point $t+1$, etc. By the definition of the ideal profile, each job of $\mathcal{J}_{(t,u)}$ is assigned to a time point in $\{t,\ldots,u-1\}$.

Proposition 1. *Edge (t,u) of G is good if and only if $\sigma_{(t,u)}$ satisfies the GHC for the ideal profile p of edge (t,u). Moreover, for any $u \in N \setminus \{0\}$, there exists a maximal good edge (t,u) in G.*

Example 2. Consider the previous example. Figure 2 (a) depicts the good edges of G. The red and blue edges are the maximal good edges of G. Among them, the red edges constitute a path from node 0 to node d. Moreover, observe that $(5,8)$ is a good edge of G, while $(6,8)$ is not. Part (b) of the same figure depicts an ideal schedule for edge $(3,7)$. The ideal profile of edge $(3,7)$ is $p_{(3,7)} = (3,2,2,2)$, since $\alpha_{(3,7)} = |\mathcal{J}_{(3,7)}|/(7-3) = 2.25$.

Proposition 2. *If edge (t,u) of G is good, then for any convex cost function, any ideal schedule of (t,u) has the least cost among all feasible schedules of $\mathcal{J}_{(t,u)}$ in the time interval $[t,u]$.*

3 The main algorithm

The algorithm *Optimize* seeks a path P in G consisting of maximal good edges only. In Step 3a, a maximal good edge is sought which ends in u. Such a maximal good edge exists by Proposition 1. In Step 3b, path P is extended with the edge (t,u) from the left.

Algorithm *Optimize*

1. If $r+1 > d$, then exit (no feasible solution exists).
2. Let $P =$ *empty path*, $u = d$.
3. While $u \geq d-r$ do:
   (a) Find the smallest $t \in \{0\} \cup \{d-r,\ldots,u-1\}$ such that edge (t,u) is good by using Proposition 1.
   (b) Let $P := (t,u) + P$. Compute the ideal schedule of interval $[t,u]$. Set $u = t$.

4. Output the schedule S^{alg} obtained by concatenating the schedules of consecutive edges of path P.

Example 3. Continuing our previous example, Algorithm Optimize determines P as the red edges of Figure 2 (a). It outputs the schedule depicted in Figure 3.

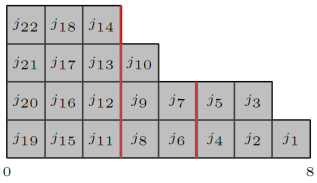


Fig. 3. Schedule obtained by Algorithm Optimize.

Theorem 2. *Algorithm* Optimize *provides an optimal solution of the resource leveling problem on a set of in-trees with a convex cost function in* $O(r^3 + r \cdot n \log n)$ *time.*

4 Computational evaluation

To better appreciate our polynomial time method, we have also formalized our resource leveling problem in a mixed-integer program with a quadratic objective function $f(\ell) = \ell^2$. When solving this model, the solver (FICO Xpress) struggled with proving optimality of the solutions found even for 20 jobs. We have also reformulated the objective function as a piecewise linear function, and then we could solve problems with up to 70 jobs within a second, but for 200 jobs the solution time was more than 7 seconds, and for 400 jobs it was more than 3 seconds. In contrast, our dedicated algorithm solves these instances in a split of a second.

Acknowledgements

This project has received funding from the European Union's Horizon Europe programme under grant agreement No. 101137954. The authors would like to thank the rest of the BATTwin consortium for supporting this research.

References

F. Ballestín, C. Schwindt and J. Zimmermann, 2007, "Resource leveling in make-to-order production: modeling and heuristic solution method", *International Journal of Operations Research*, Vol. 4(1), pp. 50-62.

P. Bendotti, L. Brunod-Indrigo, P. Chrétienne and B. Escoffier, 2024, "Resource leveling: complexity of a unit execution time two-processor scheduling variant and related problems", *Journal of Scheduling*, Vol. 27(6), pp. 587-606.

S. Hartmann and D. Birskorn, 2022, "An updated survey of variants and extensions of the resource-constrained project scheduling problem", *European Journal of Operational Research*, Vol. 297(1), pp. 1-14.

T.C. Hu, 1961, "Parallel sequencing and assembly line problems", *Operations Research*, Vol. 01, pp. 1-5.

S. Kreter, J. Rieck and J. Zimmermann, 2014, "The total adjustment cost problem: Applications, models, and solution algorithms", *Journal of Scheduling*, Vol. 17(2), pp. 145-160.

N. Megow, R.H. Möhring and J. Schulz, 2011, "Decision support and optimization in shutdown and turnaround scheduling", *INFORMS Journal on Computing*, Vol. 23(2), pp. 189-204.

Improving Earned Duration Management (EDM) Forecasting Accuracy Using a CHAID-Based Classification Framework

Amin Azimi[1], Mario Vanhoucke[123]

[1] Ghent University, Belgium
Amin.aziminasrabad@ugent.be
[2] Vlerick Business School, Belgium
[3] University College London,UK
mario.vanhoucke@ugent.be

Keywords: Project Duration Forecasting, Earned Duration Management (EDM), CHAID Regression

1 Introduction

Earned Duration Management (EDM) is a project duration forecasting approach that measures project progress directly in duration units, fully decoupling time from cost to avoid the inconsistencies of traditional EVM schedule indicators [Khamooshi and Golafshani, 2014]. Empirical studies show that EDM provides a more reliable assessment of schedule performance and can match or even outperform advanced EVM-based forecasting methods [Batselier and Vanhoucke, 2015]. Within EDM, three forecasting methods are distinguished by how the remaining duration is scaled when calculating the time Estimate at Completion (EAC(t)). EDM1 bases its forecasts directly on the planned schedule, while EDM2 and EDM3 adjust the remaining duration using the Schedule Performance Index (SPI) and the Schedule Cost Index (SCI), respectively. Furthermore, recent research shows that the forecasting accuracy of these EDM methods depends on project characteristics such as network topology, resource flexibility, and time–cost correlation [Khamooshi et al., 2021]. Given the three EDM methods and their sensitivity to project characteristics, selecting the most appropriate method for a specific project remains challenging.

To address this challenge, we propose a novel methodology to determine the most suitable EDM method for each project class. To achieve this, the study employs the Chi-squared Automatic Interaction Detector (CHAID) regression [Kass, 1980], a transparent tree-based classification technique designed to identify the project characteristics that most strongly influence forecasting accuracy. As a tree-based model, CHAID evaluates project characteristics and performs splits, where each split represents a statistically significant division of the data based on the variable most closely associated with the target. Through this sequential partitioning, the algorithm creates nodes that group structurally similar projects. The process continues until terminal nodes are generated, each representing a final class of projects with highly homogeneous characteristics.

Building on this classification, we apply a node-specific uplift adjustment inspired by Reference Class Forecasting (RCF). In RCF, an uplift is defined as a data-driven correction term that accounts for the typical forecasting deviation observed in a comparable reference class of past projects [Flyvbjerg and COWI, 2008]. Following the same principle, for each terminal node in the CHAID tree, the signed percentage forecasting errors of the projects within that structural class are collected, and their median is computed as the uplift factor. This median-based uplift quantifies the characteristic bias of that project class—indicating whether its forecasts tend to be systematically optimistic or pessimistic. Applying this class-specific uplift to the predicted completion time yields an adjusted and more realistic EAC(t) estimate for any new project assigned to that cluster.

The objective of this research is to develop a interpretable forecasting framework that: (i) classifies projects into homogeneous structural groups using CHAID regression; (ii) selects the most suitable EDM method per group; (iii) quantifies class-specific forecasting error as an uplift factor; and (iv) improves EDM duration predictions through targeted uplift adjustments.

2 Methodology

The analysis consists of four main stages. In the first stage, a large set of artificial data is generated. In the second stage, a CHAID regression model is trained to identify the characteristic profiles associated with different forecasting behaviours. In the third stage, a node-specific uplift factor is derived from the median forecasting error of similar projects and incorporated into the selected EDM method. In the fourth stage, the quality of the CHAID clustering and the effectiveness of the uplift adjustments are evaluated. Each of these four stages is explained in detail in the following paragraphs.

In the first stage, a large set of static and dynamic data is generated using artificial project instances introduced by [Vanhoucke et al., 2008] to ensure sufficient variation in project characterization. The static data consist of project characteristics available before the project start, including network-related indicators (Coefficient of Network Complexity (CNC), Order Strength (OS), Serial/Parallel indicator (SP), Activity Distribution (AD), Long Arcs (LA), Topological Float (TF)) and resource-related indicators (Resource Factor (RF), Resource Use (RU), Resource Constrainedness (RC), Resource Strength (RS)). For a detailed discussion of these indicators, readers are referred to [Vanhoucke and Coelho, 2021]. The dynamic data consist of simulated project execution and follow the framework proposed by [Ünsal Altuncan and Vanhoucke, 2025]. More precisely, each project is scheduled using a priority-rule–based approach under the Earliest Start Schedule (ESS), after which activity durations are sampled under three uncertainty settings (low, medium, and high) using lognormal distributions calibrated with different scale parameters. During the simulations, project progress is monitored and EAC(t) is computed at three tracking periods (early, middle, and late stages) using the three EDM methods individually.

Forecast accuracy is assessed using the Mean Percentage Error (MPE). At the end of this stage, the complete dataset is randomly divided into a training set (75%) and a test set (25%) for subsequent modelling and evaluation.

In the second stage, a CHAID regression model is applied to the training set to identify how project characteristics shape the forecasting behaviour of the EDM methods. Using EDM performance at the three tracking periods as the target variables, and the indicators and uncertainty levels introduced earlier as predictors, CHAID selects the predictor that best differentiates forecasting accuracy and recursively partitions the data. This process continues until no additional meaningful splits remain. The resulting terminal nodes represent distinct structural classes, and for each class the EDM method with the lowest median absolute forecasting error is selected as the best-performing method.

In the third stage, node-specific uplift factors are computed. For each node c, all projects assigned to that node provide their signed percentage forecasting errors from the selected EDM method. The median of these deviations defines the uplift factor u_c, representing the typical adjustment needed for projects in that structural class.

In the fourth stage, the CHAID classification and uplift factors are evaluated using the test set. Each project is assigned to its corresponding terminal node based on its characteristics, and its EDM estimate is corrected using the adjustment formula $\widehat{EAC}(t)_i^{adj} = \widehat{EAC}(t)_i(1+u_c)$. The accuracy improvement gained by integrating CHAID-based clustering with node-specific uplifts is then assessed by comparing adjusted forecasts with realised durations.

3 Preliminary Results

Applying the CHAID regression model to the training dataset resulted in 28 terminal nodes, each representing a structurally homogeneous class of projects. For every node, the characteristic forecasting deviation is captured by its uplift factor u_c, and the best EDM method is selected based on the lowest median absolute forecasting error. Each project assigned to node c receives an uplift-adjusted forecast $\widehat{EAC}_i(t)^{adj}$, and the resulting accuracy is re-evaluated.

Table 1. Summary of representative CHAID nodes and uplift adjustments.

Stage	**Best Method**	**Node**	**N**	**Uplift**	**Rule**	ΔErr (%)
Late	EDM1	21	30	-0.005	OS $>$ 0.63, RU $>$ 3.30, RS $>$ 0.55	4.16
Early	EDM2	24	207	0.013	OS $>$ 0.79, LA $>$ 0.07, RU $>$ 1.90	3.25
Late	EDM2	27	68	-0.026	U $>$ 0.50, SP $>$ 0.33, OS $>$ 0.45	2.51
Middle	EDM1	25	31	-0.339	U $>$ 0.50, SP $\leq$ 0.33, TF $\leq$ 0.17	6.82
Late	EDM3	22	46	0.004	U $>$ 0.50, RU $>$ 3.90, RC $>$ 0.37	8.29

Table 1 summarises a representative subset of these nodes, reporting their tracking stage, selected EDM method, uplift factor, dominant structural charac-

teristics, and the improvement in mean absolute forecasting error after applying the uplift.

The CHAID results show that measures related to network complexity frequently appear in the upper splits of the tree, indicating that structural complexity plays a major role in differentiating forecasting behaviour. Resource-related characteristics also contribute meaningfully to the model, though their influence varies across branches.

The preliminary results show that node-specific uplift factors systematically reduce forecasting bias and improve accuracy across a wide range of structural classes. Across the representative nodes analysed, the uplift adjustment leads to improvements of approximately 2%–8% in mean absolute percentage error. Although the magnitude of improvement varies across project types, the findings indicate that combining structural segmentation with uplift correction can support more adaptive and context-sensitive duration forecasting.

References

[Batselier and Vanhoucke, 2015] Batselier, J. and Vanhoucke, M. (2015). Evaluation of deterministic state-of-the-art forecasting approaches for project duration based on earned value management. *International Journal of Project Management*, 33(7):1588–1596.

[Flyvbjerg and COWI, 2008] Flyvbjerg, B. and COWI (2008). *Reference Class Forecasting for Public Projects*. Cambridge University Press.

[Kass, 1980] Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 29(2):119–127.

[Khamooshi and Golafshani, 2014] Khamooshi, H. and Golafshani, H. (2014). Edm: Earned duration management, a new approach to schedule performance management and measurement. *International Journal of Project Management*, 32(6):1019–1041.

[Khamooshi et al., 2021] Khamooshi, H., Mamghaderi, M., and Kwak, Y. H. (2021). Project duration forecasting: A simulation-based comparative assessment of earned schedule method and earned duration management. *Journal of Modern Project Management*, 9(2):7–21.

[Vanhoucke and Coelho, 2021] Vanhoucke, M. and Coelho, J. (2021). An analysis of network and resource indicators for resource-constrained project scheduling problem instances. *Computers & Operations Research*, 132:105260.

[Vanhoucke et al., 2008] Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., and Tavares, L. V. (2008). An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, 187(2):511–524.

[Ünsal Altuncan and Vanhoucke, 2025] Ünsal Altuncan, I. and Vanhoucke, M. (2025). Duration forecasting in resource constrained projects: A hybrid risk model combining complexity indicators with sensitivity measures. *European Journal of Operational Research*.

Enhancing continuous-time MILP models for resource-constrained project scheduling with workload-based constraints

Nicklas Klein, Norbert Trautmann

Department of Business Administration, University of Bern
nicklas.klein@unibe.ch; norbert.trautmann@unibe.ch

Keywords: Project scheduling, Resource-constrained project scheduling, Mixed-integer linear programming, Lower bounds.

1 Introduction

An increasing share of value-adding activities in companies is nowadays organized as projects. A key component of planning such projects is project scheduling, i.e., the allocation of scarce resources to project activities over time. A technique that has gained importance for solving project scheduling problems in recent years is mixed-integer linear programming (MILP), driven by substantial improvements in solver performance and the convenient off-the-shelf availability of modern MILP solvers for practitioners (cf. Artigues *et al.* (2015)).

We consider the resource-constrained project scheduling problem (RCPSP), which can be described as follows: given is a set of precedence-related activities that require time and capacities of scarce resources for execution. The resources are renewable, i.e., a constant number of units of each resource type (e.g., personnel or equipment) is available in each time period, and each activity requires a prescribed amount of each resource type from its start until its completion. Sought is a schedule with a minimum project completion time (makespan) that respects the precedence relations and resource availabilities.

Various approaches to formulating the RCPSP as a MILP model have been proposed in the literature. These models are typically categorized into discrete-time (DT) and continuous-time (CT) models. In DT models, the planning horizon is divided into equal-length intervals, with binary variables indicating whether an activity starts at the beginning of an interval (cf. Pritsker *et al.* (1969)). In CT models, the start times of the activities are represented via continuous variables, and binary variables are used to indicate completion-start sequencing relations between pairs of activities; different approaches have been proposed to formulate the resource constraints: the resource-flow model represents resource usage as a flow in a corresponding network (cf. Artigues *et al.* (2003)), and the resource-assignment model assigns individual resource-units to activities (cf. Rihm and Trautmann (2017)). When comparing the linear programming (LP) relaxations of DT and CT models, those of CT models are weaker than those of DT models (cf., e.g., Artigues *et al.* (2015)).

In this abstract, we review the sequence-based CT model presented in Klein *et al.* (2024), which uses binary start-start sequencing variables in addition to the completion-start sequencing variables. To strengthen the model's LP relaxation, we propose to add workload-based constraints, which can be added to other CT models as well. Our computational results on established benchmark instances indicate that the proposed constraints considerably reduce the MIP gap of the analyzed CT models.

The remainder of this abstract is structured as follows. Section 2 formally describes the RCPSP and presents an example. Section 3 reviews the sequence-based model and introduces the workload-based constraints. Section 4 presents the results of our computational study. Section 5 provides concluding remarks.

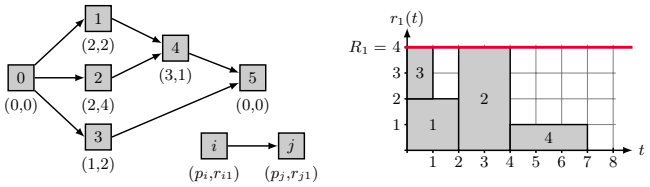
2 Problem statement

We denote the set of project activities as $V = \{0, 1, \ldots, n, n+1\}$, which is composed of n real activities and the two fictitious activities 0 and $n+1$, representing the project start and completion, respectively. The activity durations are denoted by p_i, where we assume $p_i \in \mathbb{Z}_{\geq 0}$, and the planning horizon is denoted as T. Set $E \subset V \times V$ denotes the precedence relations among pairs of activities, and TE denotes the transitive closure of E. Set R denotes the set of resource types; R_k units of type k are available, and activity i requires r_{ik} units for execution. As illustrative example, we consider a project that comprises $n = 4$ real activities, which require units of one renewable resource $k = 1$ with a capacity of $R_1 = 4$. The durations and resource requirements of the activities as well as the precedence relations among them are depicted in the activity-on-node graph in Fig. 1a.

The goal is to devise a schedule, i.e., start times of all activities, that minimizes the makespan while respecting all precedence relations and ensuring that the total requirement of each resource type does not exceed its availability at any point in time. Fig. 1b displays an optimal schedule for the example project.

Fig. 1: Example project

(a) Activity-on-node graph (b) Optimal project schedule



3 Model formulation and enhancement

In Subsection 3.1, we review the continuous-time formulation presented in Klein *et al.* (2024). In Subsection 3.2, we present the novel workload-based constraints.

3.1 Sequence-based model

The model is based on three groups of variables. The first group are continuous variables S_i, which represent the start times of the activities $i \in V$. In the example schedule in Fig. 1b, activity 2 starts at time $S_2 = 2$. The second group are binary completion-start sequencing variables y_{ij}, where $y_{ij} = 1$ indicates that activity i must be completed before activity j starts; e.g., $y_{12} = 1$ implies that activity 1 must be completed before activity 2 starts. The third group are binary start-start sequencing variables z_{ij}, where $z_{ij} = 1$ indicates that activity i starts before or at the same time as activity j; e.g., activities 1 and 3 start at the same time, i.e., $z_{13} = 1$ and $z_{31} = 1$. The sequence-based model then reads as follows:

$$\text{Min. } S_{n+1} \tag{1}$$

$$\text{s.t. } S_i + p_i \leq S_j + T(1 - y_{ij}) \qquad (i, j \in V,\ i \neq j) \tag{2}$$

$$S_j - S_i + 1 \leq T z_{ij} \qquad (i, j \in V,\ i \neq j) \tag{3}$$

$$y_{ij} = 1, \quad y_{ji} = 0 \qquad ((i, j) \in TE) \tag{4}$$

$$r_{ik} + \sum_{j \in V,\ j \neq i} r_{jk}(z_{ji} - y_{ji}) \leq R_k \qquad (i \in V,\ k \in R) \tag{5}$$

The objective (1) is to minimize the project duration, i.e., the start time of fictitious activity $n + 1$. Constraints (2) link the completion-start sequencing and the start time variables, i.e., if $y_{ij} = 1$, $S_i + p_i \leq S_j$ must hold. Constraints (3) link the start time and

start-start sequencing variables, i.e., if $S_i \leq S_j$, $z_{ij} = 1$ must hold. Constraints (4) ensure that the precedence relations are respected. Constraints (5) ensure that at the start of activity i, the total requirement of resource type k, i.e., the requirement of activity i plus the total requirement of all activities in progress at the start of activity i, does not exceed the capacity R_k. To improve the model's computational performance, additional valid inequalities can be added (cf. Klein *et al.* (2024)).

3.2 Model enhancement

The workload wl_{ik} of an activity i for resource type k is defined as its cumulated resource requirement throughout its duration, i.e., $wl_{ik} := p_i r_{ik}$. In the example project, activity 2 requires 4 resource units for 2 units of time, i.e., its workload is $wl_{21} = 8$. It is well known that the total workload of resource type k, divided by its capacity R_k, represents a lower bound on the makespan: $S_{n+1} \geq \lceil \frac{1}{R_k} \sum_{i \in V} p_i r_{ik} \rceil$. For the example project, the total workload of all activities is 17, implying a minimum project duration of $\lceil \frac{17}{4} \rceil = 5$, which equals the critical path-based lower bound, see Fig. 2a.

As an enhancement of the sequence-based model, we propose to add the following constraints that evaluate the workload processed before an activity:

$$S_j \geq \frac{1}{R_k} \Big(\sum_{i \in V,\ i \neq j} p_i r_{ik} y_{ij} \Big) \quad (j \in V,\ k \in R) \tag{6}$$

$$T y_{ij} \geq S_j - (S_i + p_i) + 1 \quad (i, j \in V, i \neq j) \tag{7}$$

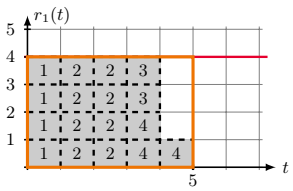
For each activity j, $y_{ij} = 1$ indicates that activity i must be completed before activity j starts, which implies that the total workload of all activities i with $y_{ij} = 1$ must be processed before the start of activity j. This is enforced by constraints (6). In the example project, the precedence relations prescribe that activity $j = 4$ cannot start before the workload of activities 1 and 2 has been processed. Since this requires at least 3 units of time, constraint (6) increases the lower bound on the project duration to 6, see Fig. 2b.

In addition to the precedence relations, completion-start sequencing relations can also be enforced via branching on some of the variables y_{ij}, which may be necessary to respect the resource constraints. To enforce that for any pair of activities i and j with $S_i + p_i \leq S_j$, $y_{ij} > 0$ must also hold, we propose to add constraints (7). In the example project, the solver may branch such that $y_{32} = 1$ holds, which implies that the workload of activities 1, 2, and 3 must be processed before the start of activity 4. In the LP relaxation of the sequence-based model, constraints (7) cause y_{34} to also take a positive value, enhancing the lower bound on the makespan in this branch to 6.1875, see Fig. 2c.

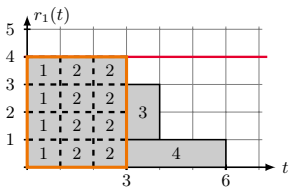
The proposed constraints are related to the concept of energetic reasoning, in which the workload is evaluated over predefined time intervals (cf., e.g., Baptiste and Demassey (2004)). The proposed constraints differ, however, in that they provide explicit lower bounds for activity start times based on precedence relations and branching decisions.

Fig. 2: Workload-based bounds for the example project

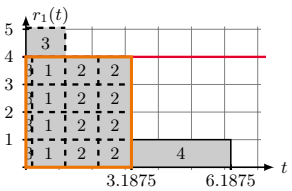
(a) Workload bound on makespan



(b) Enhanced workload bound on S_4



(c) Enhanced workload bound on S_4 after branching



4 Computational results

We used the set CV (cf. Coelho and Vanhoucke (2020)), which comprises 623 "hard to solve" instances with 20–30 activities and 1–4 resource types. We compared the sequence-based model (SEQ) against the DT model (PDT) and the CT models based on flows (FCT) and assignments (CTAB). For each model, we tested a variant without and with the novel workload-based constraints (–WL). We used an Apple workstation with an M4 Max CPU and 128 GB RAM. We implemented all models in Python 3.13, and used Gurobi 12.0.3 using up to two threads as the MILP solver with a time limit of 60 seconds per instance.

Table 1 lists the number of instances which have been solved to feasibility and optimality, the number of instances for which the best solution among all variants has been devised, the average gap between the best upper and lower bounds devised within the time limit (MIP gap), and the average gaps of the best upper and lower bounds to the critical path-based lower bound. For all CT models, our results indicate that the workload-based constraints improve their performance considerably and strongly reduce the average MIP gap. One possible reason for this is that the optimal makespan of the CT models' root-node LP relaxation increases by over 90%, on average. For the DT model, however, analogous constraints were not advantageous in this test.

Table 1: Computational results for set CV

	# Feas	# Opt	# Best	Gap^{MIP}	$\text{Gap}^{\text{UB*–CPM}}$	$\text{Gap}^{\text{LB*–CPM}}$
PDT	**623**	28	293	**12.64%**	151.04%	**119.20%**
PDT–WL	**623**	23	264	13.15%	150.87%	118.01%
FCT	11	0	0	65.33%	291.00%	2.51%
FCT–WL	386	0	0	47.65%	341.30%	108.86%
CTAB	**623**	17	131	38.66%	159.80%	55.25%
CTAB–WL	**623**	16	166	17.41%	154.94%	110.40%
SEQ	**623**	**42**	267	32.60%	151.69%	66.19%
SEQ–WL	**623**	40	**404**	14.39%	**148.58%**	112.32%

5 Conclusions

In this abstract, we presented workload-based additional constraints that can be added to MILP models for the RCPSP. Our computational results indicate that these additional constraints enhance the performance of CT models. For future research, we suggest to adapt the constraints to variants of the RCPSP. Moreover, symmetrical constraints should be analyzed that evaluate the workload processed after the completion of an activity.

References

Artigues C., P. Michelon and S. Reusser, 2003, "Insertion techniques for static and dynamic resource-constrained project scheduling", *European Journal of Operational Research*, Vol. 149(2), pp. 249–267.

Artigues C., O. Koné, P. Lopez and M. Mongeau, 2015, "Mixed-integer linear programming formulations", In: C. Schwindt, J. Zimmermann (eds) *Handbook on Project Management and Scheduling Vol.1*. Springer, Cham.

Baptiste, P., S. Demassey, 2004, "Tight LP bounds for resource constrained project scheduling", *OR Spectrum*, Vol. 26, pp. 251—262.

Coelho J., M. Vanhoucke, 2020, "Going to the core of hard resource-constrained project scheduling instances", *Computers & Operations Research*, Vol. 121, 104976.

Klein N., M. Gnägi and N. Trautmann, 2024, "Mixed-integer linear programming for project scheduling under various resource constraints", *European Journal of Operational Research*, Vol. 319(1), pp. 79–88.

Pritsker A., L. Waiters and P. Wolfe, 1969, "Multiproject scheduling with limited resources: a zero-one programming approach", *Management Science*, Vol. 16(1), pp. 93–108.

Rihm T., N. Trautmann, 2017, "An assignment-based continuous-time MILP model for the resource-constrained project scheduling problem", *Proceedings of the 2017 IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 35–39.

An interaction-oriented action decision model for project control under budget constraints

Yuxuan Song[1] and Mario Vanhoucke[1,2,3]

[1] Faculty of Economics and Business Administration, Ghent University, Belgium
yuxuan.song@ugent.be, mario.vanhoucke@ugent.be
[2] Technology and Operations Management, Vlerick Business School, Belgium
[3] UCL School of Management, University College London, United Kingdom

Keywords: Project control, Risk interaction, Corrective action decision.

1 Introduction

Project control is performed iteratively in each project phase to identify project problems and take appropriate corrective actions to bring the project back on track. In the literature, corrective action decision is broadly approached from two fundamentally different perspectives. The first stream of research is developed based on the project network (PN) consisting of activities with precedence relations, where uncertainty in activity durations (or costs) is modeled using probability distributions unrelated to the actual sources of risks. Two types of corrective actions are typically defined for the PN, i.e., *activity crashing* that reduces the duration of activities and *variability reduction* that reduces the variability (activity risk) of activities. A second stream of research focuses on the so-called risk interaction networks (RIN), which consist of risk events with interconnections, forming a connected profile of the various disruptions that could impact project objectives. In these studies, two types of corrective actions are defined for the RIN, i.e., *risk prevention* that reduces the occurrence probabilities of risks and *risk protection* that mitigates the impacts of risks. Both research streams can be further categorized into *metric-based* and *optimization-based* action decision approaches. Specifically, the metric-based approach identifies a number of critical activities or risks on which corrective actions should be taken, whereas the optimization-based approach constructs mathematical models to determine an optimal set of corrective actions that best achieve various project objectives.

To the best of our knowledge, only two papers address the integration of such risk interaction networks with project networks for project control [Song and Vanhoucke, 2025,Chen et al., 2023], and this is also a central theme in our current research. In the study of Chen et al. (2023), critical risks are identified under risk interactions to support the development of a pre-project risk management plan. However, their work focuses on static risk analysis and does not consider dynamic action decisions during project execution. In the study of Song and Vanhoucke (2025), a double-layer network model (abbreviated as PN-RIN) is constructed, and sensitive activities are identified. Given that the effort the project manager puts in project control is limited, the *control effort* is defined to determine the optimal number of sensitive activities that might be

subject to corrective actions [Vanhoucke, 2010]. Three control strategies are proposed to determine the corrective actions during the project execution, namely a *preventive strategy* (which focuses on future sensitive activities), an *interventive strategy* (which focuses on ongoing sensitive activities), and a *hybrid strategy* (which focuses on both ongoing and future sensitive activities).

This study extends the existing work from two perspectives. First, in practice, the project manager always set aside additional budget to implement corrective actions, and the amount of available budget is limited. In this study, we establish a unified framework to support the selection and implementation of three existing control strategies under limited budgets. Second, the existing control strategies are developed based on the metric-based action decision approach. In this study, we construct a mathematical model to formulate the action decision problem and develop an algorithm to solve it. Extensive computational experiments are conducted to compare the performance of different control strategies.

2 Problem description

2.1 PN-RIN model

A double-layer network (PN-RIN) is constructed by linking a project network (PN) and a risk interaction network (RIN) to model the interaction between the risks and the activities of a project, as visualized in Figure 1. More precisely, a project is initially represented by an activity-on-the-node network consisting of activities between finish-to-start precedence relations, namely the project network. Then, the project risks and the interactions between these risks (cause/effect relations) are identified and quantified as probabilities, forming the risk interaction network, which is similar to the project network. Since each risk affects the duration of a single activity, the final step involves connecting the relationships between activities in the project network and risks in the risk interaction network, representing the time delays in activity duration caused by the occurrence of the corresponding risk.

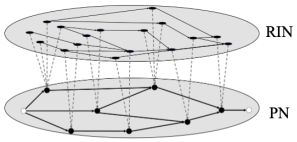


Fig. 1: The PN-RIN model

2.2 Control strategies

Four types of corrective actions are generally defined for the project network and the risk interaction network. In this study, the RIN causes uncertainty in activity durations, so the action of reducing the variability in the probability distributions of activity durations [Vaseghi et al., 2024] can be interpreted as

risk prevention within the RIN. Furthermore, since risk impacts in the PN-RIN are modeled as increases in activity durations, *risk protection* can be represented as activity crashing in the PN. Accordingly, two corrective actions are explicitly defined in the PN-RIN framework: activity crashing for the PN and risk prevention for the RIN.

Based on these two actions, four project control strategies are developed labelled as preventive, interventive, hybrid, and optimization-based. The *preventive* strategy focuses solely on risk prevention, applying actions to future risks in the RIN according to their ranking by the risk criticality index until the allocated budget is exhausted or no eligible risks remain. The *interventive* strategy applies activity crashing to ongoing activities in the PN, guided by the activity sensitivity index and constrained by budget or action availability. The *hybrid* strategy combines both approaches, sequentially applying risk prevention and activity crashing until the budget is fully utilized. These three strategies are all drawn from the literature and clearly follow a metric-based approach in which critical activities or risks are identified using the activity sensitivity index and the risk criticality index. In contrast, the *optimization-based* strategy formulates corrective action selection as a mathematical optimization problem, determining under a limited budget the optimal combination of activity crashing and risk prevention that minimizes total project cost while ensuring the expected project duration meets the predefined deadline.

3 Methodology

To compare different control strategies, a simulation model is constructed, distinguishing three separate phases. **Phase 1 (Static Simulation)** involves calculating activity sensitivity metrics and risk criticality metrics based on the PN-RIN model. **Phase 2 (Dynamic Simulation)** simulates the project execution through successive tracking periods (TPs) until the project completion. At the start of the project, the tracking period is TP = 0 and it is subsequently increased (TP = TP + 1) until the project is completely finished. During each tracking period, a portion of the total control budget B is allocated as the allowable budget consumption for period TP (B_{TP}). Corrective actions are then selected and implemented according to the chosen control strategy, subject to the budget constraint. Ultimately, at the end of the dynamic simulation runs, many possible project executions have been simulated (with corrective actions), and the performance of the project control system will then be measured in **Phase 3 (Performance measurement)** using the time effectiveness (TE) metric originally proposed by [Martens and Vanhoucke, 2019] and the cost efficiency (CE) metric proposed by [Song et al., 2020].

4 Results

Since the experiments for the optimization-based strategy are still in progress, we currently present the comparison results of the preventive, interventive, and hybrid strategies under four different scenarios. Specifically, the tracking periods were set at different points in time according to the percentage of project

completion (PC), which is measured by dividing the earned value at a given point in time by the total project budget. The four scenarios are summarized as shown in Table 1.

As shown in Figure 2, the hybrid strategy consistently outperforms the interventive strategy and preventive strategy, highlighting the importance of designing actions not only for solving current problems but also future expected problem. Furthermore, the time effectiveness of using the interventive strategy depends on both the timing and the quantity of control periods, while the time effectiveness of using the preventive strategy and the hybrid strategy primarily depend on the timing of control periods.

The detailed computational experiment results will be shown at the conference.

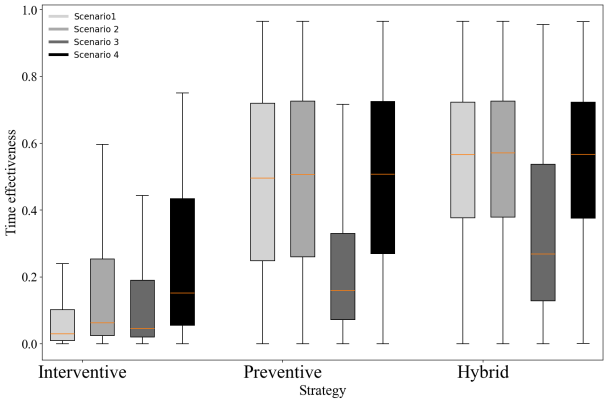


Fig. 2: Comparison results under different control periods

Table 1: Different scenarios for tracking periods

	Scenarios			
	1	2	3	4
20% PC	✓	✓		✓
40% PC		✓		✓
60% PC			✓	✓
80% PC			✓	✓

References

[Chen et al., 2023] Chen, L., Lu, Q., and Han, D. (2023). A bayesian-driven monte carlo approach for managing construction schedule risks of infrastructures under uncertainty. *Expert Systems with Applications*, 212:118810.

[Martens and Vanhoucke, 2019] Martens, A. and Vanhoucke, M. (2019). The impact of applying effort to reduce activity variability on the project time and cost performance. *European Journal of Operational Research*, 277(2):442–453.

[Song et al., 2020] Song, J., Martens, A., and Vanhoucke, M. (2020). The impact of a limited budget on the corrective action taking process. *European Journal of Operational Research*, 286(3):1070–1086.

[Song and Vanhoucke, 2025] Song, Y. and Vanhoucke, M. (2025). Schedule risk analysis for project control with risk interactions. *Annals of Operations Research*, pages 1–56.

[Vanhoucke, 2010] Vanhoucke, M. (2010). Using activity sensitivity and network topology information to monitor project time performance. *Omega*, 38(5):359–370.

[Vaseghi et al., 2024] Vaseghi, F., Martens, A., and Vanhoucke, M. (2024). Analysis of the impact of corrective actions for stochastic project networks. *European Journal of Operational Research*, 316(2):503–518.

B - Constraint programming #1

Parallel machine scheduling for PCB assembly with precedence, setup times and uncertain machine speeds: a proactive-reactive approach

Youssef Karouma[1,2,3], Christian Artigues[2], Houssem-eddine Gharbi[1], Le Toan Duong[1], et Romain Guillaume[3]

[1] Schaeffler group, Toulouse, France
{youssef.karouma, le.toan.duong, houssem-eddine.gharbi}@vitesco.com
[2] LAAS-CNRS, Universite de Toulouse, CNRS, Toulouse, France
{youssef.karouma, christian.artigues}@laas.fr
[3] IRIT, Universite de Toulouse, Toulouse, France
{Youssef.Karouma, Romain.Guillaume}@irit.fr

Mots-Clefs. uniform machine scheduling, printed circuit board assembly, time-varying uncertain machine speed, scheduling decision tree

1 Introduction and presentation of the problem

We consider a parallel machine scheduling problem, where the inputs are a set of jobs $\mathcal{J} = \{J_1, ..., J_n\}$ with release dates $r_1, \ldots, r_n$, due dates $d_1, \ldots, d_n$ and a set of machines $\mathcal{M} = \{P_1, ..., P_m\}$, and the output is a schedule, which assigns each task J_i to a machine P_k, and gives its start time S_i. The time that machine P_k requires to process job J_i is denoted by $p_{i,k}$. Each job J_i has to be scheduled by one machine out of a set $M_i \subseteq \mathcal{M}$. One of the important variants is uniform-machine scheduling. In this problem, some machines are uniformly faster than others in processing each job. This means that each machine has a speed factor v_k which impacts the run time of a job J_i on it, given a nominal job duration p_j, as follows: $p_{i,k} = p_i/v_k$.
In the context of electronic manufacturing, and especially Surface-Mount technology (SMT), plants have automated and continuous production lines, (identified here to machines) which operate 24-7. These lines follow complex processes, in which each PCB (printed circuit board) undergoes many transformations through several steps. The production process goes generally through 4 big phases : Front-End, ICT(In-Circuit Testing), Back-end and Packaging. In the present work, we're interested in Front-end, which consists in assembling electronic devices on a PCB board via SMT placement machines. More precisely, during front-end, each PCB has 2 jobs to execute, mounting devices on its 1st face and mounting devices on its 2nd face. This yields a precedence constraint between the 2 jobs. Consequently, the static and deterministic form of our problem is a strongly NP-hard uniform-machine scheduling problem denoted $Q|r_i, M_i, s_{i,j,k}, prec|\sum T_i$ in the standard 3-field notation for scheduling. Setup time $s_{i,j,k}$ is a delay needed before starting a new job J_j on a machine P_k, knowing that we had a job J_i processed before on the same machine.
In practice, the line speed varies over time, and can be defined as a stochastic function of time. For a particular scenario the speed function is denoted $v_k(t)$. Then on this scenario, if a task is assigned to line k, its start time S_j and its completion time C_j satisfy $\int_{t=S_j}^{t=C_j} v_k(t)dt = p_j$.

2 Constraint programming model to solve the deterministic form of the problem

2.1 Decision variables

- $x_{i,k} \in \{0, 1\}$: if job i is assigned to line k.

- $s_i \geq 0$: start time of job i.
- $e_{i,j,k} \in \{0, 1\}$: if job j succeeds job i on line k.
- $[s_i, s_i + p_{i,k}]$: interval of job i if assigned to line k.
- $C_i \geq 0$: completion time of PCB i after face 2.
- $T_i \geq 0$: tardiness of PCB i.

2.2 Constraints

- **Precedence between faces**: $s_{i,face_2} \geq s_{i,face_1} + p_{i,face_1}$
- **AddNoOverlap** constraint: no overlap between intervals $[s_i, s_i + p_i]$ on the same line.
- **setup time** $set_{i,j,k}$ between 2 consecutive jobs on the same line.
- **Completion time of a job i if assigned to line k**: $C_i = s_i + p_{i,k}$ where $p_{i,k} = \frac{p_i nominal}{\bar{v_k}}$, $\bar{v_k} = \frac{1}{T} \int_{t=0}^{t=T} v_k(t)dt$.

2.3 Objective

- $min \sum_i T_i$ where $T_i = max(0, C_{i,face_2} - d_{i,face_2})$.

3 Proactive approach to handle uncertainty: scheduling decision tree

Indeed, lines speeds evolution over time are unknown beforhand. They evolve in a way we can't perfectly predict. Hence, jobs durations $p_{i,k}$ are not determined at the beginning, which makes the problem harder to solve. The figure below shows 2 completely different scenarios of lines speeds evolution over time.

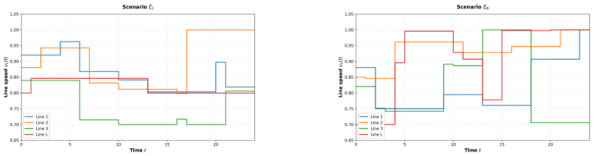


Fig. 1. Example of 2 scenarios for lines speeds evolution over time

To handle efficiently the uncertainty, we propose a proactive approach: a scheduling decision tree. The idea is that at the beginning, we compute an initial schedule S_0 based on the average scenario. Then at a checkpoint time t_θ, we observe what is actually happening; i.e : the partial scenario until now; and we classify it as either *Good* or *Bad* for S_0. If it's *Good*, we keep S_0. If it's *Bad*, we trigger a reschedule and compute a new schedule S_1. This schedule takes into account the current state of S_0 execution following the partial scenario until t_θ, and for the future the average of the *Bad* scenarios. This approach is illustrated in the following formula:

$$S_{\pi_\theta}(\xi) = \begin{cases} S_0, & \text{if } \mathrm{Good}_\theta(\xi[0, t_\theta]) \\ \mathrm{S}_1\Big(t_\theta,\ S_0(\xi_1[0, t_\theta]),\ \widehat{\Xi}_{\mathrm{bad\_for\_}S_0}[t_\theta, T]\Big), & \text{if } \mathrm{Bad}_\theta(\xi[0, t_\theta]) \end{cases} \tag{1}$$

- $\xi_1[0, t_\theta]$ is a partial possible scenario between 0 and t_θ.
- $S_0(\xi_1[0, t_\theta])$ means the current status of schedule S_0 on partial scenario $\xi_1[0, t_\theta]$ at time t_θ (finished tasks/ongoing tasks/pending tasks).

– $\widehat{\Xi}_{\text{bad\_for\_}S_0}[t_\theta, T]$ means the average scenario on Bad classified scenarios over t_θ.

In fact, the minimization problem could be stated as follows:

$$\min_{\theta \in \Theta} \frac{1}{|\Xi_{\text{train}}|} \sum_{\xi \in \Xi_{\text{train}}} P(S_{\pi_\theta}(\xi), \xi)$$

We want to minimize the average performance over all training scenarios, where performance is the sum of tardinesses. The parameter θ controls the classifier that partitions scenarios into *Good* and *Bad* for S_0. As solving directly this problem could be very complex, we decided to handle it in a progressive way.

4 Learn the rescheduling checkpoint t_θ and condition $condition_\theta$

To learn t_θ and the classification rule $condition_\theta$, we follow a training procedure.

4.1 Training phase

First, we sample a training data of scenarios Ξ_{train}, and compute schedule S_0 for the average scenario $\hat{\Xi}_{train}$. Then, for each scenario ξ, we use a simulator to obtain the label of ξ towards S_0 : $label = sim(S_0, \xi)$. We consider that the label is *Good* if at least 40% of products meet their due dates, and *Bad* otherwise. Once we have the whole labeled dataset, we train a classification tree on time features $\{v_{line_1}(t = t_i), ...v_{line_L}(t = t_i)\}$. Finally, a checkpoint time and a classification rule $(t_\theta, condition_\theta)$are outputted.

4.2 Learning the earliest reliable rescheduling checkpoint

When training a standard classificatiion tree, it could output $condition_\theta$ using late-time features (e.g: t=23). Hence, rescheduling will be triggered too late to be effective. Consequently, we use the algorithm below in order to ensure a trade-off between good detection of bad scenarios and early anticipation time t_θ. The main idea of this algorithm is to train a classifier on the whole time features of scenarios dataset $\{v_k(t)\}_{t<=t_{max}}$ such that $t_{max} = T$ (total horizon). At each step, we monitore the quality the tree and check if it satisfies a certain condition. If true, we reduce t_{max} by the half and continue. Else, we stop at the final obtained tree, trained using the smallest possible t_{max}.

4.2.1 Dichotomy-based feature reduction algorithm

Algorithm 1: Dichotomy-based feature reduction

Input: Training set Ξ_{train} with labels w.r.t. S_0, maximum horizon T
Output: Checkpoint time t_θ and classification rule $condition_\theta$

$t_{max} \leftarrow T$;
$tree^* \leftarrow \emptyset$;
while $t_{max} > 0$ **do**
  Train classification tree $\mathcal{T}$ on features $\{v_k(t)\}_{t \leq t_{max}}$;
  if *quality*($\mathcal{T}$) *satisfies criterion* **then**
    $tree^* \leftarrow \mathcal{T}$;
    $t_{max} \leftarrow \lfloor t_{max}/2 \rfloor$;
  else
    break;
return $t_\theta = t_{max}$, $condition_\theta$ *from* $tree^*$

where the quality *criterion* ensures a sufficient detection of Bad scenarios.

5 Experimental protocol to evaluate the approach

We sample 13 instances of training set scenarios. For each instance, $condition_\theta$ splits Ξ_{train} into $\Xi_{classified\_bad\_for\_S_0}$ and $\Xi_{classified\_good\_for\_S_0}$. We want to assess if the rescheduling transforms some scenarios in $\Xi_{classified\_bad\_for\_S_0}$ to being *good* for S_1. Thus, for each $\xi \in \Xi_{classified\_bad\_for\_S_0}$, we compute the new schedule S_1 as described in 1. Then, we simulate S_0 and S_1 on ξ, and we see if an initial *bad* label for S_0 becomes *good* for S_1. While repeating this for all the subset of bads, we can compute the rate of bad scenarios for S_0 converted into good for S_1.

$$\text{Rate}_{\text{bad\_for\_}S_0 \Rightarrow \text{good\_for\_}S_1} = \frac{|\{\xi \in \Xi_{\text{Classified\_Bad\_for\_}S_0} \mid \text{LabelS}_0(\xi) = \text{Bad and LabelS}_1(\xi) = \text{Good}\}|}{|\Xi_{\text{Classified\_Bad\_for\_}S_0}|} \tag{2}$$

The left side of the figure below represents this metric on the 13 training instances. Overall, it stays above 50% in the most cases, which means that our approach manages to turn initially bad scenarios for S_0 into good ones for S_1. Moreover, we evaluate our rescheduling approach on a test set instance (distinct seed from training). Across 15 random scenarios ξ from this set, we simulate S_0 and S_1 and compare their sum of job tardinesses ($P(S_0(\xi), \xi)$ and $P(S_1(\xi), \xi)$). As shown in the right side of the figure below, rescheduling reduces tardiness in $11/15$ scenarios (73%), with an average improvement $P(S^0(\xi), \xi) - P(S^1(\xi), \xi) = 2.6$, confirming the effectiveness of our approach.

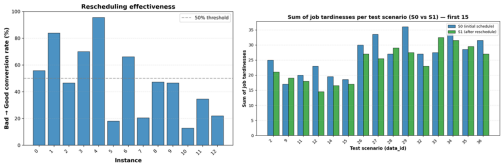


Fig. 2. the left side shows the rate of scenarios transformed from bad towards S_0 into good towards S_1. The right side shows improvement of objective function (sum of jobs tardinesses) across 15 random test scenarios

6 Conclusion and future work

To summarize, we proposed an hybrid approach that learns when and how to reschedule using a classification tree trained on scenario data. We showed it can reliably detect bad situations early and improve the scheduling outcome. By repeating the same procedure for S_1 (sampling scenarios, train a classifier, output a checkpoint and rescheduling condition $(t_\theta, condition_\theta)$), we can obtain the whole scheduling tree with nodes S_0, S_1, S_2... as well as conditions guiding decisions until all tasks are completed. We will also study alternative criteria and variants of interpretable learners to balance robustness, tree depth, and re-optimization cost.

References

T. Portoleau, C. Artigues, R. Guillaume, 2024. "Robust decision trees for the multi-mode project scheduling problem". *European Journal of Operational Research*, vol. 312(2), pp. 525–540.

T. Portoleau, C. Artigues, R. Guillaume. "Robust Predictive-Reactive Scheduling: An Information-Based Decision Tree Model", 2021.

Resource-Constrained Assembly Line Balancing Problem with Learning Effect and Fixed Adjustment Periods: A CP Approach

Duc Anh Le[1], Stéphanie Roussel[1] and Christophe Lecoutre[2]

[1] ONERA/DTIS, Université de Toulouse, France
duc_anh.le@onera.fr, stephanie.roussel@onera.fr
[2] CRIL-CNRS, Université d'Artois, France
christophe.lecoutre@cril.fr

Keywords: Resource-Constrained Assembly Line Balancing Problem, Learning Effect, Ramp-up, Constraint Programming.

1 Introduction

The ramp-up phase, during which the production process is repeatedly executed until the target production rate is reached, is a major challenge for many industries. This paper addresses the ramp-up of a pulsed aircraft assembly line, where the whole aircraft – or a subsystem – is assembled across a series of workstations. Each workstation processes the aircraft for a fixed *cycle time*, and the aircraft must pass through all workstations sequentially. Designing such a line can be modeled as a Resource-Constrained Assembly Line Balancing Problem (RC-ALBP), requiring task allocation and scheduling within workstations while respecting resource limits (aircraft zones, machines, operators, etc.) and minimizing the cycle time [Ağpak and Gökçen, 2005].

We focus on the *learning effect*, *i.e.* the reduction in assembly-task durations that occurs throughout the ramp-up stage as tasks are repeated [Grosse et al., 2015]. This effect results from continuous improvements in the aircraft production system processes, supply chain coordination, factory flows, operator skills, etc., while successive aircraft are built on the line. In earlier work, we introduced the RC-ALBP with Learning (RC-ALBLP/L), whose goal is to design a resource-constrained assembly line that accounts for learning effect [Le et al., 2025]. The problem consists of assigning tasks and resources to each workstation, scheduling the tasks, and selecting a cycle-time value for each aircraft entering the line (called a *period*) so as to minimize two criteria: (1) the ramp-up duration required to reach a given target cycle time, and (2) the number of cycle-time adjustments made during the ramp-up. In fact, since each adjustment can cause delays and extra costs, reducing their number is desirable. This paper addresses a restricted version, assuming that the candidate adjustment periods are predetermined and examining the consequences of this assumption. We present the corresponding CP model and experiments on realistic industrial data, showing that this restriction (which computationally simplifies the problem) significantly speeds up computation compared with [Le et al., 2025].

2 CP Encoding

Problem Inputs. The inputs of the problem, RC-ALBLP/L with fixed period adjustments, are:

- the target cycle time c_{tgt} and the maximum possible cycle time c_{max},
- the number of workstations W and the associated set $\mathcal{W} = [\![1, W]\!]$,
- the number of periods P and the associated set $\mathcal{P} = [\![1, P]\!]$, where each period corresponds to the entry of a new examplary on the line. $\mathcal{P}^A$ is the subset of periods for which an adjustment is allowed, i.e. for which it is possible to modify the line cycle time. Periods of $\mathcal{P}^A$ are denoted p_1, p_2 …, p_n, so that p_i is the period where the i-th adjustment is done. $\mathcal{P}^U$ is the subset of *unstable* periods containing the first W periods during which the line is being filled. We assume that $\mathcal{P}^A \cap \mathcal{P}^U = \emptyset$. For compactness, we note $\mathcal{P}^C = \mathcal{P}^A \cup \mathcal{P}^U$.
- the set of resources $\mathcal{R}$, where each resource r models a limited pool of operators or machines that have to be dispatched on the line,
- the set of aircraft zones $\mathcal{Z}$ in which operators perform assembly tasks, where each zone has a limited capacity,
- the set of tasks $\mathcal{T}$ to be performed on the line, with for each task t its monotonically decreasing duration function $dur_t(k)$ that depends on the number k of times it has already been completed. A task might occupy or neutralize resources and zones during its execution.

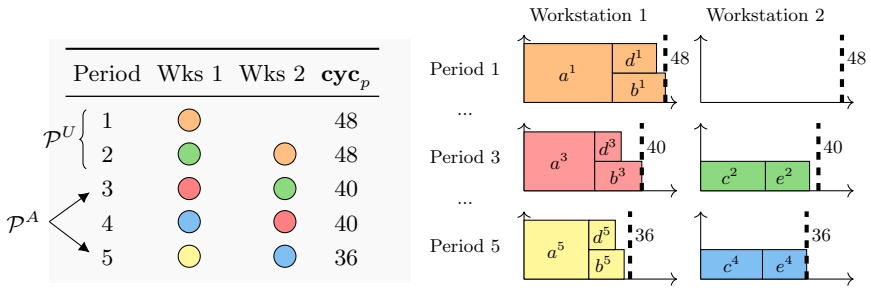


Fig. 1: Small instance of the problem with 2 workstations, 5 periods and 5 tasks.

Example. Figure 1 illustrates a toy instance with 2 workstations and 5 periods, including 2 unstable periods ($\mathcal{P}^U = \{1, 2\}$) and 2 adjustment periods ($\mathcal{P}^A = \{3, 5\}$). Each colored circle denotes a product unit under assembly. At the end of each cycle time a new period begins, units advance to the next workstation, and a fresh unit enters the line. A single resource r (capacity 3) is allocated between the workstations. The instance comprises five tasks (a, b, c, d, e); task a consumes two units of r, while the others consume one. The target cycle time is 36, and precedence and zone constraints are omitted for simplicity. The right part of Figure 1 shows a solution for the toy problem. Workstation 1

receives two units of resource r and hosts tasks a, b and d; workstation 2 receives one unit of r and hosts tasks c and e. Thanks to the learning effect, task durations shrink over periods – for example, a^3 (the third occurrence of a) is shorter than a^1. Because a cycle-time adjustment is allowed in period 3, the cycle time is set to 40, the minimum feasible value for both stations. The learning effect similarly enables the target cycle time to be reached by period 5. Even if not presented here, the tasks scheduling can be modified within workstation if it allows to reach a smaller cycle time value during an adjustment period.

Decision Variables. In what follows we omit the classic RC-ALBP task precedence and resource allocation variables and constraints, which are identical to those in [Le et al., 2025] (integer variables for modelling resource allocation, precedence and cumulative resource constraints on interval variables). We instead concentrate on the novel aspects of the paper: the learning effect and the predetermined adjustment periods. The decision variables are introduced next.

- For each task $t \in \mathcal{T}$, the integer variable $\mathbf{wks}_t \in \mathcal{W}$ indicates the workstation to which t is allocated. Note that the workstations allocation does not depend on the period, i.e. it cannot be changed during ramp-up.
- For each task $t \in \mathcal{T}$, each period $p \in \mathcal{P}^C$ and each workstation $w \in \mathcal{W}$, the optional interval variable $\mathbf{itv}_{t,p,w} \in [\![0, c_{max}]\!]$ represents the execution of task t during the period p at the workstation w. This variable is present only if t is assigned to w.
- For each period $p \in \mathcal{P}^C$, the integer variable $\mathbf{cyc}_p \in [\![c_{tgt}, c_{max}]\!]$ indicates the cycle time value decided for period p.

Criteria & Constraints

$$\textit{minimize} \quad (p_1 - 1) \cdot \mathbf{cyc}_1 + \sum_{i=2}^{n} (p_i - p_{i-1}) \cdot \mathbf{cyc}_{p_{i-1}} + \mathbf{cyc}_{p_n} \tag{1}$$

$$\forall t \in \mathcal{T}, \forall p \in \mathcal{P}^C, \forall w \in \mathcal{W}, \qquad \mathsf{presenceOf}(\mathbf{itv}_{t,p,w}) = \mathsf{equal}(\mathbf{wks}_t, w) \tag{2}$$

$$\forall p \in \mathcal{P}^U, \qquad \mathbf{cyc}_p = \max_{t \in \mathcal{T}, q \in \mathcal{P}^U, w \in \mathcal{W}} \mathsf{endOf}(\mathbf{itv}_{t,q,w}) \tag{3}$$

$$\forall p \in \mathcal{P}^A, \qquad \mathbf{cyc}_p \geq \max_{t \in \mathcal{T}, w \in \mathcal{W}} \mathsf{endOf}(\mathbf{itv}_{t,p,w}) \tag{4}$$

$$\mathbf{cyc}_{p_n} = c_{tgt} \tag{5}$$

$$\forall t \in \mathcal{T}, \forall p \in \mathcal{P}^C, \forall w \in \mathcal{W}, \qquad \mathsf{lengthOf}(\mathbf{itv}_{t,p,w}) = dur_t(p - w) \tag{6}$$

The objective (Eq. 1) is to minimize the ramp-up stage duration, which spans from the beginning of production until the end of last adjustment period. The constraints (2) link the presence of interval variables with the allocation of tasks. Constraints (3), (4) and (5) compute the cycle time for each period (we suppose that there is no cycle time adjustment during the unstable periods). Note that, in Constraints (4), $\mathbf{cyc}_p$ cannot take a value smaller than c_{tgt}, resulting in an inequality. Constraints (6) calculate the duration for each task in each period. The number of completed executions k of a task allocated on workstation w at period p can be calculated by $p - w$.

3 Experiments

Following the flexible adjustment model of [Le et al., 2025], we tested six real-world aircraft-assembly instances, named `[#tasks]-[#workstations]`. We fix some regularly spread adjustment periods over $\mathcal{P}$, i.e. $\{5, 10, 17, 26\}$ and $\{8, 15, 22, 29\}$ for 2-workstation ($P = 26$) and 4-workstation ($P = 50$) cases, respectively. These values are arbitrary and correspond to realistic adjustement periods dates with regards to operational constraints (regularity and enough time for successive adjustments). Table 1 presents the associated results: BST (Best Solution Time) is the CPU time (seconds) to obtain the best solution. The restricted formulation reduces the problem size by a factor of around 4 (2-workstation) and 6 (4-workstation), yielding dramatically faster solution times while delivering near-optimal – or even optimal – solutions, something the flexible $\mathcal{P}^A$ approach cannot guarantee. Although $\mathcal{P}^A$ is predetermined, the ramp-up durations remain comparable to the original model and are better in some cases. Any feasible solution of the restricted model is also feasible for the original problem, though optimality may be lost. Nevertheless, when solution quality drops with the fixed $\mathcal{P}^A$, the flexible version requires substantially more time to reach the same objective value. The `199-2` instance is an exception regarding the BST value; however, the restricted model finds a solution of $91,461$ in 22 seconds, whereas the original model needs $1,278$ seconds to achieve the same value.

One of the many perspectives of this work is to extend the problem so that any adjustement on the line results in a duration penalty on activities. It would also be possible to consider some uncertainty on the learning effect.

Instance	Fixed $\mathcal{P}^A$		Flexible $\mathcal{P}^A$ [Le et al., 2025]		
	Ramp-up duration	BST	Ramp-up duration	BST	Decided $\mathcal{P}^A$
`187-2`	$\mathbf{93,876}$	8	$97,098$	$3,315$	$\{3, 5, 9, 16\}$
`199-2`	$90,506$	$2,588$	$\mathbf{85,311}$	$1,484$	$\{3, 4, 6, 10\}$
`795-2`	$93,672$	120	$\mathbf{90,316}$	$4,216$	$\{3, 5, 8, 14\}$
`187-4`	$\mathbf{71,937}$*	10	$77,988$	$3,687$	$\{5, 8, 17, 31\}$
`199-4`	$85,244$*	10	$\mathbf{82,094}$	$1,507$	$\{5, 10, 21, 32\}$
`795-4`	$\mathbf{68,486}$	20	$78,085$	$6,568$	$\{6, 8, 16, 24\}$

Table 1: Computational impact of fixed and flexible period adjustments

References

[Ağpak and Gökçen, 2005] Ağpak, K. and Gökçen, H. (2005). Assembly line balancing: Two resource constrained cases. *International Journal of Production Economics*, 96(1):129–140.

[Grosse et al., 2015] Grosse, E., Glock, C., and Müller, S. (2015). Production economics and the learning curve: A meta-analysis. *International Journal of Production Economics*, 170:401–412.

[Le et al., 2025] Le, D. A., Roussel, S., and Lecoutre, C. (2025). Aircraft resource-constrained assembly line balancing with learning effect: A constraint programming approach. In *CP 2025*, volume 340 of *LIPIcs*, pages 25:1–25:24.

Multi-valued decision diagram integration method applied for a specific parallel machine problem

Hugo Apeloig[1], Odile Bellenguez[1],
Guillaume Massonnet[1] and Gilles Simonin[1]

Atlantique, LS2N, UMR CNRS 6004, 44307, Nantes, France
hugo.apeloig@imt-atlantique.fr, odile.bellenguez@imt-atlantique.fr,
guillaume.massonnet@imt-atlantique.fr, gilles.simonin@imt-atlantique.fr

Keywords: constraint programming, multi-valued decision diagram, scheduling, unrelated parallel machine, setups, renewable resources.

1 Introduction

This work is originally motivated by the industrial scheduling problem defined in [Carvin, 2025]. Here, we focus on a sub-problem; the Unrelated Parallel Machine scheduling problem with renewable Resource consumption and machine and sequence dependent Setup times (UPMSR). Resource consumption is considered during processing time as well as setup operations. Resources can be specific to a type of operation (*e.g.* machines) or shared (*e.g.* employees). This sub-problem can be seen as a UPMSR [Fanjul-Peyro, 2020] and is described later on. To address this problem, we propose to adapt a new method introduced in [Apeloig et al., 2026], recalled in Section 1, and adapted in a novel constraint programming (CP) model in Section 2.

UPMSR problem Unrelated Parallel Machine scheduling problem (UPM) is defined over a set of tasks $\mathcal{J}$ and machines $\mathcal{W}$. Processing time of task j can vary from one machine to another and is parameterized by p_{jm}, for all $m \in \mathcal{W}$. In our case, we consider the *minimization* of the *makespan* (*i.e.* total completion time). The problem is extended with mandatory *machines* and *sequence-dependent* setup times. Setups are given by the matrix σ, with σ_{ijm} being the duration of the setup that must be processed between task i and task j on m. Note that σ_{ijm} can be different from σ_{jim}. As in [Fanjul-Peyro, 2020], we consider that the triangular inequality must be satisfied. Moreover, a setup must be executed right before the execution of its corresponding task. Additional resources are *renewable* and can be sometimes called *secondary resources* in opposition to the machine (*principal* resources). Secondary resources are depicted by the tuple $(\mathcal{R}, \mathcal{B}, b)$, where $\mathcal{R} = \{R_1, \ldots, R_q\}$ denotes the q secondary resources, and availabilities are noted with $\mathcal{B} = \{B_1, \ldots, B_q\}$. In UPMSR, setup operations also consume secondary resources. The demand is defined over two matrices, b^p for tasks and b^s for setups. b^p_{jmk} depicts the quantity of resource k

needed to process task j on machine m. The demand of resource k for the setup of task j preceded by i on machine m is noted b^s_{ijmk}. Finally the overall problem is defined over the tuple $(\mathcal{J}, \mathcal{W}, p, \sigma, \mathcal{R}, \mathcal{B}, b)$. A valid solution is a schedule that also respects, for all t, the constraint $\sum_{j \in A_t} \sum_{m \in \mathcal{W}} b_{jmk} \leq B_k$, for each resource k with A_t being the set of active tasks and setups at t. In other words, at any time period t the *cumulative* usage of all tasks and setups in progress must not exceed the total capacity.

CP and MDD-Global-Constraints This type of problem is typically addressed with CP models using cumulative constraints [Baptiste et al., 2001]. The Cumulative constraint considers multiple tasks and ensures that at any time, the total demand of a renewable resource k is no more than B_k. In [Apeloig et al., 2026], it is also demonstrated how advantageous it could be to capture features of tasks in a multi-valued decision diagram (MDD) directly into the cumulative. A MDD can be constructed from a set of variables as a rooted, directed acyclic graph that has a layer for each variable [Castro et al., 2022]. In this method, a MDD is defined for each task and each path corresponds to a valid mode. Results were provided for the MDD-Multi-Cumulatives constraint where it enabled stronger communication between constraints. It filters starting and completion times as well as machine (or mode) assignment of tasks and it is based on the timetabling algorithm proposed in [Gay et al., 2015]. The two main principle of this method are (i) to encode objects with MDDs that can be query to obtain minimal or maximal values of variables with respect to certain assumptions, and (ii) MDDs are embedded as parameters inside global constraints, enabling a stronger filtering from the same set of assumptions already used by the constraint. So we proposed to apply and extend this approach for UPMSR, using choco solver [Prud'homme and Fages, 2022]. All other global constraints used in this work are formally defined in [Beldiceanu et al., 2010].

2 Constraint programming modelling

In the following, variables are denoted in uppercase. Given $i \in \mathcal{J}$ we define two vectors $\mathcal{X}^p_i$ and $\mathcal{X}^s_i$ respectively for the task and its setup. Variable Y_i denotes the machine on which i is processed and E_i is its direct predecessor. Let $\mathcal{X}^p_i = \langle Y_i, S^p_i, P^p_i, C^p_i, H^p_{ik} | \forall k \in 1, \ldots, \mathcal{R} \rangle$, where S^p_i, P^p_i and C^p_i correspond to the starting time, the duration and the completion time. Finally, H^p_{ik} is the demand of resource k. In the same way, for the setup we have $\mathcal{X}^s_i = \langle Y_i, E_i, S^s_i, P^s_i, C^s_i, H^s_{ik} | \forall k \in 1, \ldots, \mathcal{R} \rangle$. Note that as the setup exactly ends when the task starts, the variable S^p_i can be used as the completion time of the setup (*i.e* $C^s_i = S^p_i$). Finally, H^s_{ik} is the demand of resource k. Given $i \in \mathcal{J}$, we define two task variables $t^p_i = (S^p_i, P^p_i, C^p_i)$ and $t^s_i = (S^s_i, P^s_i, C^s_i = S^p_i)$. It maintains consistency over its start, duration and completion (*i.e.* $S^p_i + P^p_i = C^p_i$).

We propose as a reference a first model (called decomposed model). First, we use Element constraints to maintain consistency between tasks' duration

and demand, and the machine on which it is processed. Consider for example the duration, the corresponding constraint is $\textsc{Element}(P_i^p, [p_{im}|\forall m \in \mathcal{W}], Y_i)$. The same way, consistency is maintained for each setup. As the constraint is defined over a one-dimensional vector, we flatten the matrices. We define an auxiliary variable $Prec_i = Y_i \times n + E_i$ to correspond to the predecessor in these matrices. Consistency between the task i and its predecessor is ensured by multiple constraints. Constraint $\textsc{Element}(Y_i, [Y_j|\forall j \in \mathcal{J}], E_i)$ ensures that they are both processed on the same machine. Let F_i, an auxiliary variable, correspond to the completion time of the predecessor of i. Constraints $\textsc{Element}(F_i, [C_j^p|\forall j \in \mathcal{J}^0], E_i)$ and $F_i \leq S_i^s$ ensure that t_i^s starts after its completion. Moreover, constraints AllDifferent and AllDifferentExcept0 are added over $Prec_i$ and E_i respectively. Finally, Cumulative constraints are enforced over the overall set of task variables (tasks and setups) for each additional resource and machine.

MDD model Let $\mathcal{M}_i^p$ and $\mathcal{M}_i^s$ be the MDDs corresponding to respectively task i ($\mathcal{X}_i^p$) and its setup ($\mathcal{X}_i^s$). Consistency over duration and demand configurations is maintained with MDDC constraints in (3) and (4). As in the decomposed model, there are multiple constraints linking tasks together. (5) ensure that a task i and its predecessor j are processed on the same machine. (6) and (7)-(8) breaks any sub-cycles in the schedule. We use the MDD-Multi-Cumulatives constraint in (9) as defined in [Apeloig et al., 2026]. Finally, (1) and (2) minimize the overall completion time of the schedule.

$$
\begin{aligned}
\min \quad & C_{\max} && && (1)\\
\text{s.t} \quad & \textsc{Max}(C_{\max}, [C_i^p, \forall i \in \mathcal{J}]) && && (2)\\
& \text{MDDC}(\mathcal{X}_i^p, \mathcal{M}_i^p) && ,\forall i \in \mathcal{J} && (3)\\
& \text{MDDC}(\mathcal{X}_i^s, \mathcal{M}_i^s) && ,\forall i \in \mathcal{J} && (4)\\
& \textsc{Element}(Y_i, [Y_j|\forall j \in \mathcal{J}], E_i) && ,\forall i \in \mathcal{J} && (5)\\
& \textsc{AllDifferent}([E_i|\forall i \in \mathcal{J}]) && && (6)\\
& \textsc{Element}(F_i, [C_j^p|\forall j \in \mathcal{J}^0], E_i), && \forall i \in \mathcal{J} && (7)\\
& F_i \leq S_i^s && ,\forall i \in \mathcal{J} && (8)\\
& \text{MDD-}\textsc{Multi-Cumulative}(\mathcal{X}, \mathcal{M}, B) && && (9)\\
& t_i^p = (S_i^p, P_i^p, C_i^p), t_i^s = (S_i^s, P_i^s, S_i^p) && ,\forall i \in \mathcal{J} && (10)\\
& 0 \leq F_i \leq S_i^s \leq S_i^p \leq C_i^p, && ,\forall i \in \mathcal{J} && (11)\\
& 1 \leq Y_i \leq |\mathcal{W}| && ,\forall i \in \mathcal{J} && (12)\\
& 0 \leq E_i \leq n \wedge E_i \neq i && ,\forall i \in \mathcal{J} && (13)
\end{aligned}
$$

Setup Specific MDDs $\mathcal{M}_i^s$ is augmented in order to consider problem-specific restrictions when it is queried. Consider a query made to obtain the minimal value of any variable of $\mathcal{M}_i^s$ with respect to the assumption $Y_i = m$. This assumption restricts that i and its setup are executed on m. It also implies that

the predecessor j of i will be executed on m. As j can belong to $\mathcal{D}(E_i)$ as long as it exists at least one machine conjointly available for both, it may exist m and j such that $j \in \mathcal{D}(E_i)$ and $m \notin \mathcal{D}(Y_j)$. Thus, when identifying feasible paths of $\mathcal{M}_i^s$ with respect to $Y_i = m$, the condition $E_i = j \implies m \in \mathcal{D}(Y_j)$ must also hold. This restriction avoid unfeasible path and speed up the resolution process.

3 Preliminary results and discussions

We test our models on the instances proposed in [Fanjul-Peyro, 2020] for which there are at most three additional resources, one only for the task, one for the setup and one shared. We also extend these benchmarks by testing over instances with more than one shared resource. Preliminary numerical experiments show that the decomposed CP model obtains results comparable to the state-of-the-art mixed-integer programming (MIP) presented in [Fanjul-Peyro, 2020]. As we present a generic CP model, we only compare it to the MIP model and not the three-stage process. The MDD approach provides even better results with a reduction of up to 90% of the number of nodes needed to prove optimality. Nevertheless, the side of an MDD depends on the variable ordering. Thus, a focus will be made in order to observe which arrangement reduces the size depending on the instances characteristics. In a next step, such methods will be used to address more complex industrial problems such as FCPSP [Carvin, 2025].

Acknowledgments. This publication has emanated from research conducted with the Tandem financial support of Region Pays de La Loire.

References

[Apeloig et al., 2026] Apeloig, H., Beldiceanu, N., Bellenguez, O., Massonnet, G., and Simonin, G. (2026). Mdd integration for global constraints: The multi-cumulatives case. Technical report.

[Baptiste et al., 2001] Baptiste, P., Le Pape, C., and Nuijten, W. (2001). *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media.

[Beldiceanu et al., 2010] Beldiceanu, N., Carlsson, M., and Rampon, J.-X. (2010). Global constraint catalog.

[Carvin, 2025] Carvin, B. (2025). *Optimisation d'un ordonnancement de production de produits périssables dans un contexte multi-lignes*. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique.

[Castro et al., 2022] Castro, M. P., Cire, A. A., and Beck, J. C. (2022). Decision diagrams for discrete optimization: A survey of recent advances. 34(4):2271–2295.

[Fanjul-Peyro, 2020] Fanjul-Peyro, L. (2020). Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. 5:100022.

[Gay et al., 2015] Gay, S., Hartert, R., and Schaus, P. (2015). Simple and scalable time-table filtering for the cumulative constraint. In Pesant, G., editor, *Principles and Practice of Constraint Programming*, pages 149–157. Springer International Publishing.

[Prud'homme and Fages, 2022] Prud'homme, C. and Fages, J.-G. (2022). Choco-solver: A java library for constraint programming. *Journal of Open Source Software*, 7(78):4708.

Logic-based Benders Method for Multi-Skill Resource-Constrained Project Scheduling with Allocation and Skill Flexibility

Carla Juvin[1], Christian Artigues[2] and Pierre Lopez[2]

[1] TBS Business School, Toulouse, France
c.juvin@tbs-education.fr
[2] Univ Toulouse, CNRS, LAAS, Toulouse, France
christian.artigues, pierre.lopez@laas.fr

Keywords: Scheduling, Skills, Logic-based Benders decomposition, Mixed-integer programming, Constraint programming.

1 Introduction

In this paper, we study a multi-skill resource-constrained project scheduling problem (Snauwaert & Vanhoucke 2023), where we assume that a worker can cover several skills and the worker allocation can change during the processing of activities (Polo Mejía 2019). We propose an exact logic-based Benders decomposition (LBBD) method (Hooker 2000) whose master problem is a scheduling problem, where the allocation of workers is relaxed into cumulative resource constraints. This master problem is solved by Constraint Programming. The subproblems consist in verifying, using Integer Linear Programming, that worker allocation is possible at any instant. Logical feasibility cuts are issued from the subproblems in the form of new cumulative resource constraints.

2 Problem definition

The multi-skill resource-constrained project scheduling problem with allocation and skill flexibility (MS-RCPSP-ASF) considers a set of activities $\mathcal{A} = \{1, \ldots, n\}$ and a set of cumulative resources $\mathcal{CR}$ with limited capacity B_k, $\forall k \in \mathcal{CR}$. We also consider a set $\mathcal{L} = \{1, \ldots, L\}$ of skills, and a set $\mathcal{HR}$ of disjunctive resources that represent multi-skilled workers. A parameter $m_{w,\ell} \in \{0, 1\}$ indicates whether worker $w \in \mathcal{HR}$ masters skill $\ell \in \mathcal{L}$ ($m_{w,\ell} = 1$) or not ($m_{w,\ell} = 0$). Each activity $i \in \mathcal{A}$ is defined by a processing time $p_i \in \mathbb{N}$, a demand $b_{i,k} \in \mathbb{N}$ for each cumulative resource $k \in \mathcal{CR}$, and a requirement $a_{i,\ell} \in \mathbb{N}$ for each skill $\ell \in \mathcal{L}$. In addition, processing an activity i requires a minimum number of workers q_i, independently of the required skills. Furthermore, there is a set E of precedence constraints, where $(i, j) \in E$ means that activity j cannot start before activity i is completed. Finally, our objective in MS-RCPSP-ASF is to find a schedule that minimises the total duration of the project or makespan ($C_{\max}$). The MS-RCPSP-ASF is NP-hard as it includes the RCPSP as a particular case.

3 Logic-based Benders Method

The MS-RCPSP-ASF involves both scheduling and assignment decisions. The structure of the problem therefore lends itself naturally to decomposition. LBBD method is often used in this case (Cire *et. al.* 2016). When using LBBD to solve such integrated scheduling and resource allocation problems, the resource allocation part is usually modelled by the master problem (MP), while the subproblem (SP) deals with the scheduling part (Li &

Womer 2009). The MP is then generally solved by integer linear programming (ILP) and the SP by constraint programming (CP), in accordance with the predominance of these paradigms on the corresponding problems. In contrast, in our approach, scheduling is addressed in the MP via CP and then, if feasible, the workers are allocated by ILP in the SP; see also Yi-fan *et al.* (2013).

3.1 Problem Decomposition

At each iteration, the MP provides a schedule, which defines only the time periods during which each activity is processed and minimises the makespan. This gives a lower bound (LB) for the objective function. Then, the SP consists in finding an allocation of workers to activities in each time period that satisfies both capacity and requirement constraints. If such an assignment is possible, the optimal solution is reached. Otherwise, if no assignment is possible, a heuristic is used to generate a feasible solution to the problem and thus an upper bound (UB). If this UB equals the lower bound given by the MP, the solution provided by the heuristic is optimal. Otherwise, cuts are derived and added to the master problem, which is then solved again. This process is repeated until an optimal solution is found or a stopping criterion is reached. The general LBBD scheme applied to our problem is illustrated in Figure 1.

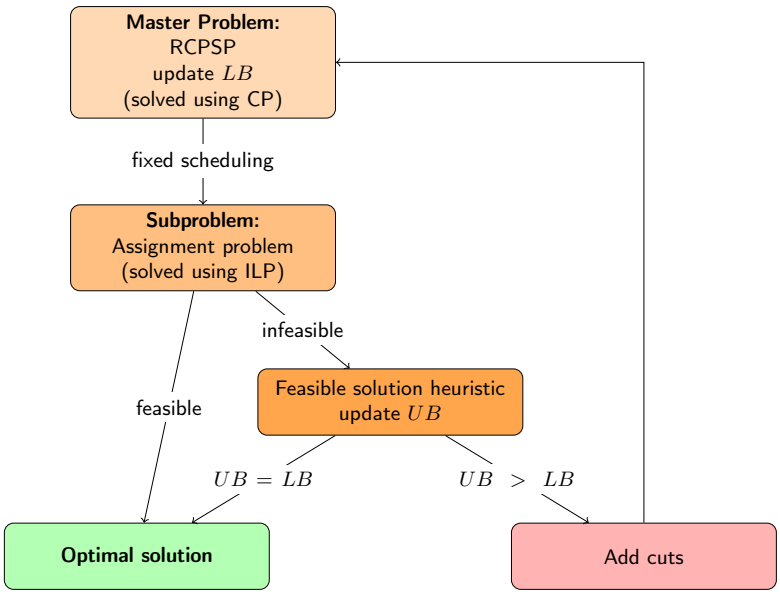


Fig. 1: Decomposition scheme for the MS-RCPSP-ASF

3.2 Master Problem

The master problem is the RCPSP relaxation of the MS-RCPSP-ASF where the allocation of workers to activities is ignored. The decision variables are therefore only the activity processing dates and the objective is to find a schedule that minimises the makespan, while satisfying precedence constraints and capacity of cumulative resources. It is modelled and solved using Constraint Programming, with a classical RCPSP model, where an interval

variable $\texttt{act}_i$ is defined for each activity $i \in \mathcal{A}$. Although the decision variables concerning the allocation of workers do not appear in this model, additional constraints are added to guide the master problem. To that purpose, skills and workers usage constraints are relaxed into cumulative resources.

3.3 Subproblems

Given that the allocation of workers can change during the processing of the activities, it suffices to make sure that there exists a feasible assignment for every time period. Thus, the subproblem can be split into a number of smaller problems. Note that if there exists a feasible assignment at a given time, this assignment remains feasible at least until another activity starts. We can therefore split the time horizon into periods we call *delta-periods*.

For a given delta-period Δ, we note $\mathcal{A}_\Delta$ the set of activities starting or in progress at the beginning of Δ. The aim of the subproblem is to find a feasible worker allocation for each activity.

The objective is to maximise the number of activities assigned, w_Δ, while ensuring that no worker is allocated to more than one activity and that the minimum number of workers and skill requirements for each activity are met. We show that, for a delta-period, the subproblem is strongly NP-hard, even when $|\mathcal{A}_\Delta| = 2$ by reduction from Set MultiCover. The subproblems are solved using Integer Linear Programming.

3.4 Logic Cuts

The subproblem gives w^*_Δ as the maximum number of activities in $\mathcal{A}_\Delta$ that can be processed in parallel. Then, when no assignment is possible, this is trivially inferred to the master problem thanks to the following cumulative constraint:

$$\sum_{i \in \mathcal{A}_\Delta} pulse(act_i, 1) \leq w^*_\Delta \tag{1}$$

where $pulse(\texttt{ivar}, h)$ is a function of time that takes the value h between the start time and the end time of the associated interval variable `ivar` and 0 otherwise. Each cut (1) only prevents more than w^*_Δ activities of $\mathcal{A}_\Delta$ from being scheduled in parallel, while there could exist other activity subsets with similar skill requirements that cannot be scheduled in parallel. In order to strengthen their impact, more general cuts are introduced.

By solving a problem formulated using integer linear programming, we determine whether there is a set $S \subseteq \mathcal{L}$ of skills whose requirements, for activities in $\mathcal{A}_\Delta$, exceed the number of available resources with these skills. In this case, we define $\mathcal{A}_\mathcal{S}$ as the set of activities that require at least one skill of set S and $\mathcal{HR}_S$ as the set of workers who master at least one of the skills in S, and add cut:

$$\sum_{i \in \mathcal{A}_\mathcal{S}} pulse(act_i, \max_{\ell \in S}(a_{i,\ell})) \leq |\mathcal{HR}_S| \tag{2}$$

As for the standard skills, the demand of all activities scheduled in parallel must not exceed the available quantity of workers possessing at least one skill in S.

4 Results and Conclusion

For computational experiments, we consider a set of 360 instances derived from classical PSPLIB RCPSP benchmark instances. These instances are extended by randomly generating skills, skilled workers, and activity skill requirements. The test set includes projects

with 30, 60, 90, and 120 activities, 8, 12, and 20 workers, and 10, 15, and 20 skills. Figure 2 reports the computational results, showing the number of optimal solutions found and the optimality gap for the different instance sizes, for the method presented in this paper and a pure CP method using CP Optimizer.

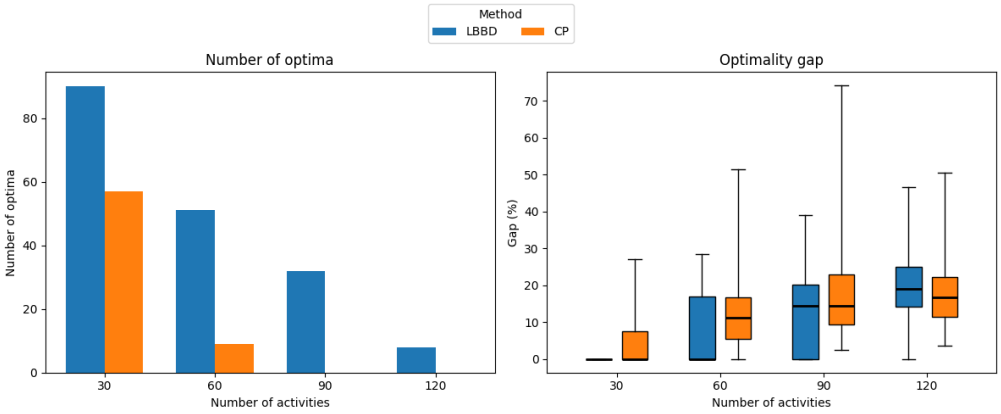


Fig. 2: LBBD and CP performance according to the number of activities

Although the proposed method outperforms the pure CP method in terms of optimal solutions found and proved, it still has difficulties in obtaining good upper bounds for difficult instances. The obtained optimality gaps are similar to the ones obtained by the pure CP method.

Consequently, future research could incorporate more efficient heuristics to find feasible solutions. It would also be interesting to study the proposed decomposition method for other variants of the multi-skill scheduling problem, particularly for the standard case, where the allocation must remain the same throughout the processing of activities and where each worker can only cover a single skill.

References

Cire, A. A., Coban, E., and Hooker, J. N., 2011, "Logic-based Benders decomposition for planning and scheduling: A computational analysis", *The Knowledge Engineering Review*, 31(5), 440–451.

Hooker, J. N., 2000, *Logic-based methods for optimization: Combining optimization and constraint satisfaction*, John Wiley & Sons, New York, 2000.

Li, H. and Womer, K., "Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm", *Journal of Scheduling*, 12(3), 281–298.

Polo Mejía, O., 2019, *Operational research approach for optimising the operations of a nuclear research laboratory*, PhD thesis, Institut National des Sciences Appliquées de Toulouse, France.

Snauwaert, J. and Vanhoucke, M., 2023, "A classification and new benchmark instances for the multi-skilled resource-constrained project scheduling problem", *European Journal of Operational Research*, 307(1), 1–19.

Yi-fan, W., Fu-quan, S., Shi-xin, L., and Di, C., 2013, "A new method to solve project scheduling problems with multi-skilled workforce constraints", In *2013 25th Chinese Control and Decision Conference (CCDC)*, pages 2408–2412. IEEE, 2013.

C - Mixed-integer linear programming

Scheduling of Star Observations under Uncertain Conditions: A Comparison of Models and Solvers

Thomas Rahab Lacroix, Pierre Lemaire, Nadia Brauner

Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, 38000 Grenoble, France
{thomas.rahab-lacroix, pierre.lemaire, nadia.brauner}@grenoble-inp.fr

Keywords: scheduling, optimization under uncertainty, robustness, min-max regret, arc flow, solver comparison.

1 Scheduling with uncertainty on the number of machines

We consider a scheduling problem for star observations using a telescope: M identical nights are available to observe J stars that have been identified as of interest by astronomers. Each star $j \in \{1, 2, ..., J\}$ yields a profit w_j if it is observed; in that case, the observation must respect a visibility window $[r_j, d_j]$ and a minimum observation duration of at least p_j. The objective is to maximize the total profit of the observations that are actually performed. By interpreting nights as machines and stars as jobs to be processed, this problem is exactly $PM|r_j|\sum w_j U_j$ in Graham's notation, a classic scheduling setting of parallel machines.

In the practical star-observation problem, the ability to observe depends on meteorological and atmospheric conditions, which are not known when the schedule is computed. A possible modeling is to consider that each night can be either perfect and everything is observable, either terrible and nothing is observable. The challenge is therefore to propose a "high-quality" schedule without knowing the actual number of nights (machines) that will ultimately be available (Stein *et. al.* 2019); this contrasts with classical scheduling models, where uncertainty typically affects the job-related data.

This problem can be viewed from the perspective of stochastic optimization when probabilities over the number of available nights are known (Rahab Lacroix *et. al.* 2025). Since such probabilities are difficult to obtain in practice, we focus here instead on evaluating the worst case within the robust optimization paradigm.

2 *Min-max regret* formulation

As the nights are identical, when one appears observable, a schedule must execute its night with the highest profit. So we can assume that all schedules have their nights sorted by decreasing profit. We denote by $\mathcal{X}$ the set of feasible schedules over M nights. The objective is therefore to propose a schedule $x \in \mathcal{X}$ computed for M nights such that for each $m = 1, ..., M$, its restriction to its first m nights is not too far from the optimal schedule over m nights. Let $x(m)$ denotes the cumulative profit of schedule x over its first m nights, and let $OPT_m \in \mathcal{X}$ be an optimal schedule over m nights for the problem without uncertainty. The difference between these two values is the regret, and the minimization of the maximum regret (Kasperski *et. al.* 2014) can then be written as (1). (1) can be rewritten as a decision problem (2) by introducing a bound R:

$$\min_{x \in \mathcal{X}} \left(\max_{m \in \{1,2...,M\}} \Big(OPT_m(m) - x(m) \Big) \right) \tag{1}$$

$$\exists x \in \mathcal{X}, \quad \forall y_1, y_2 ..., y_M \in \mathcal{X}, \quad \forall m \in \{1, 2..., M\}, \quad y_m(m) - x(m) \leq R \tag{2}$$

Indeed, for any schedule $y \in \mathcal{X}$, we have $OPT_m(m) \geq y(m)$, and there exists one for which equality holds. Therefore, the above formulation corresponds precisely to the decision version of the maximum regret minimization problem. The alternation of an "exists" and a "for all" quantifier places this problem in the complexity class Σ_2^p (Woeginger 2021). Moreover, the quantifiers can be inverted, so the problem is also in Π_2^P. Besides, the problem can be formulated as a MILP with an objective function different than just the regret, and thus belong to Δ_2^P. However, the exact complexity is not known yet.

3 Models for the computation of an optimal schedule

To compute the regret, it is essential to compute as efficiently as possible an optimal schedule over m machines, for each $m = 1, ..., M$. We consider three possible Integer Linear Programming (ILP) formulations for this scheduling problem: Start Time (ST), Time Discretization (TD), and Arc Flow (AF) and compare their performances.

The classical Start Time formulation (Wagner 1959) involves $O(J^2)$ binary variables (see Model 1). Its simplicity is balanced by poor solving capacity due to the use of big-M (here a $T = \max_j(d_j)$) constraints and the quickly increasing number of binary variables.

In the Time Discretization formulation (Sousa *et. al.* 1992), a night is discretized into $T + 1$ instants. This model involves $O(MTJ)$ binary variables (see Model 2). Its flexibility and ability to solve quite large instances is limited when the number of nights, or the number of instants per night, is too high.

The Arc Flow formulation (Kramer *et. al.* 2020) models the scheduling problem as a flow in a graph. Time is discretized into $T + 1$ instants, and the vertices correspond to the time instants from 0 to T. For each star j and each time $t \in [r_j, d_j - p_j]$, an arc from t to $t + p_j$ with weight w_j is created. Additional arcs are used to represent idle times. A path from vertex 0 to vertex T in this graph corresponds to the schedule of a single night. The objective is to find a maximum-weight flow from vertex 0 to vertex T that satisfies additional constraints: each star can be scheduled at most once, and the total flow leaving 0 is bounded by M.

The Arc Flow model involves $O(TJ)$ binary variables (see Model 3) and is considered in the literature very powerful. However this formulation cannot be used to solve other more general variants of the problem, e.g. when nights are not identical. Because there is no information about the night when an observation is scheduled, this formulation cannot be used when partial evaluations are required (e.g. robust or stochastic optimization).

4 Comparison of solvers and models

Astronomers provided us with a list of 800 stars with their profits. Then, 120 dates were selected uniformly over the 10 last years and time windows and processing times for all stars were computed for each situation. As consecutive nights are quite similar in terms of time windows, an instance over M nights is built by repeating M times the same situation (with M taking its value in $\{1, 3, 7, 14, 30, 100\}$). For each instance, we keep all visible stars (about 400 to 600 among the 800 stars). At the end, we obtain 720 distinct realistic instances.

Those different instances are solved multiple times. With the 3 models presented before: ST, TD and AF but also with 2 solvers: HiGHS and CPLEX (Saukh *et. al.* 2024). This brings it to a total of 4320 resolutions and each of them is allowed 2 minutes of computation on a server equipped with an Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz, 20 physical cores, and 40 threads.

$a_{nj} = 1$ if observation j is scheduled night n, and 0 otherwise
$b_{ij} = 1$ if observation i is scheduled before observation j the same night, and 0 otherwise
c_j = start time of observation j

$$\begin{aligned}
\text{Maximize} \quad & \sum_{n=1}^{M}\sum_{j=1}^{J} a_{nj} w_j & \\
\forall j, \quad & \sum_{n=1}^{M} a_{nj} \leq 1 & \text{observation } j \text{ is scheduled at most once} \\
\forall j, \quad & r_j \leq c_j \leq d_j - p_j & \text{observation } j \text{ is scheduled in its time window} \\
\forall i \neq j, \quad & c_i + p_i \leq c_j + T(1 - b_{ij}) & \text{ensure the no overlap of observations } i \text{ and } j \\
\forall i \neq j, n, \quad & b_{ij} + b_{ji} \geq a_{nj} + a_{ni} - 1 & \text{enforce an order for observations } i \text{ and } j \\
\forall i, j, n, \quad & a_{nj} \in \{0,1\}, b_{ij} \in \{0,1\}, & c_j \geq 0
\end{aligned}$$

Model 1. Start Time (Wagner 1959)

$s_{ntj} = 1$ if observation j starts at instant t the night n, and 0 otherwise

$$\begin{aligned}
\text{Maximize} \quad & \sum_{n=1}^{M}\sum_{j=1}^{J} w_j \sum_{t=0}^{T} s_{ntj} & \\
\forall j, \quad & \sum_{n=1}^{M}\sum_{t=0}^{T} s_{ntj} \leq 1 & \text{observation } j \text{ is scheduled at most once} \\
\forall j, n, \quad & \forall t \in [\![0, r_j - 1]\!], s_{ntj} = 0 & \text{observation } j \text{ starts after } r_j \\
\forall j, n, \quad & \forall t \in [\![d_j - p_j + 1, T]\!], s_{ntj} = 0 & \text{observation } j \text{ ends before } d_j \\
\forall t, n, \quad & \sum_{j=1}^{J} \sum_{u=t-p_j+1}^{t} s_{nuj} \leq 1 & \text{at most one observation an instant } t \\
\forall j, t, n \quad & s_{ntj} \in \{0,1\} &
\end{aligned}$$

Model 2. Time Discretization (Sousa et. al. 1992)

$f_{tj} = 1$ if observation j starts at instant t, and 0 otherwise

$$\begin{aligned}
\text{Maximize} \quad & \sum_{j=1}^{J}\sum_{t=0}^{T} f_{tj} w_j & \\
\forall j \neq 0, \quad & \sum_{t=0}^{T} f_{tj} \leq 1 & \text{observation } j \text{ is scheduled at most once} \\
\forall j, \quad & \forall t \in [\![0, r_j - 1]\!], f_{tj} = 0 & \text{observation } j \text{ starts after } r_j \\
\forall j, \quad & \forall t \in [\![d_j - p_j + 1, T]\!], f_{tj} = 0 & \text{observation } j \text{ ends before } d_j \\
\forall t, \quad & \sum_{j=0}^{J} f_{tj} - f_{(t-p_j)j} = \begin{cases} M \text{ if } t = 0 \\ -M \text{ if } t = T \\ 0 \text{ else} \end{cases} & \text{flow conservation} \\
\forall j, t, \quad & f_{tj} \in \{0,1\} \text{ except } f_{t0} \in \mathbb{R}^+ &
\end{aligned}$$

Model 3. Arc flow (Kramer et. al. 2020)

	Number of instances where optimality is proven over 120 dates						Mean time (s) to prove optimality when proven under 2min					
	1	3	7	14	30	100	1	3	7	14	30	100
ST	0/0	0/0	0/0	0/0	0/0	0/0	_/_	_/_	_/_	_/_	_/_	_/_
TD	99/120	2/120	0/3	0/0	0/0	0/0	64.7/7.6	122.9/30.4	_/108.7	_/_	_/_	_/_
AF	118/120	120/119	120/120	120/120	118/120	92/120	43.6/4.3	16.5/3.8	20.7/4.3	30.5/4.5	38.7/5.0	47.0/6.0

Table 1. Performances of the different models and solvers. Each cell contains two values that are obtained for the above metric with the alongside formulation for the above number of nights with the solvers HiGHS/CPLEX.

A clear results from Table 1 about the models is that the AF formulation is by far the best tool to solve the OPT_m problem. Moreover, with the actual characteristics of our instances, the TD formulation is strictly better than the ST formulation.

About the solvers, Table 1 shows that CPLEX is faster than HiGHS to prove optimality no matter the formulation. In average, CPLEX is 6 times faster than HiGHS. This ratio is low considering that CPLEX is a commercial solver and HiGHS an open source solver.

5 Conclusion

We presented three models to solve the $PM|r_j|\sum w_jU_j$ problem. Their performances are compared on realistically sized instances (up to 100 nights and 600 stars) using different solvers that are also compared. We conclude that to solve this problem, the Arc Flow formulation is by far the most efficient. In the perspective of solving a robust or stochastic formulation, the Time Discretization model seems more appropriate than the Start Time model. Besides, reducing the size of an instance by using only stars selected by Arc Flow for an optimal resolution is a promising heuristic.

To conclude, the next step is to add metrics to analyse the performances of the models and solvers, such as the number of instances with a gap to optimality lower than 5% and the number of instances reaching the best known solution obtained with the same model. Moreover, new commercial and open source solvers will be add to the comparison. Experiments with CBC, Gurobi, and GLPK are currently in progress.

References

Kasperski Adam and Pawel Zielinski, 2014, "Minmax (regret) scheduling problems", *Sequencing and scheduling with inaccurate data*, pp. 159-210.

Kramer Arthur, Dell'Amico Mauro, Feillet Dominique and Iori Manuel, 2020, "Scheduling jobs with release dates on identical parallel machines by minimizing the total weighted completion time", *Computers and Operations Research*, Vol. 123.

Rahab Lacroix Thomas, Lemaire Pierre, Lagrange Anne-Marie, Milli Julien and Brauner Nadia, 2025, "Scheduling Ground-Based Telescope Observations with Uncertain Nights", *Fourteenth International Workshop on Planning and Scheduling for Space (IWPSS 2025)*.

Saukh Sergii and Puchko Taras, 2024, "Mixed-Integer Linear Programming Solvers for Local Grid Capacity Planning Problems", *2024 IEEE 5th KhPI Week on Advanced Technology*.

Sousa Jorge P and Laurence A Wolsey, 1992, "A time indexed formulation of non-preemptive single machine scheduling problems", *Mathematical programming*, Vol. 54, pp. 353-367.

Stein Clifford and Zhong Mingxian, 2019, "Scheduling When You Do Not Know the Number of Machines", *ACM Transactions on Algorithms (TALG)*, Vol. 16, pp. 1-20.

Wagner Harvey M, 1959, "An integer linear-programming model for machine scheduling", *Naval research logistics quarterly*, Vol. 6, pp. 131-140.

Woeginger Gerhard J, 2021, "The trouble with the second quantifier", *4OR*, Vol. 19, pp. 157-181.

The Unreliable Job Selection and Sequencing Problem

Emmeline Perneel[1], Alessandro Agnetis[2], Roel Leus[1], and Ilaria Salvadori[2]

[1] Research Centre for Operations Research and Statistics, KU Leuven
[2] Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena

Keywords: single-machine scheduling, job selection, unreliable jobs, submodular optimization, algorithm design.

1 Problem statement

We consider a single-machine problem with unexpected breakdowns. We are given a set $J = \{1, \ldots, n\}$ of jobs. Each job has a cost $c_j \geq 0$ which must be paid if the job is selected, representing preparatory or setup expenses that must be paid regardless of whether the job is completed successfully. A reward $r_j \geq 0$ is achieved if the job is successfully carried out. The machine can fail during the execution of a job, and in that event, the job is lost, and no further job can be carried out. The probability that the machine fails during the execution of job j is given by $1 - \pi_j$, where $\pi_j \in [0, 1]$ is called the success probability of job j. If a certain subset S of jobs is selected for processing, let σ denote a sequence of these $|S|$ jobs, and let $\sigma(k)$ represent the k-th job in the sequence σ. The probability of reaching and successfully carrying out the k-th job in the sequence is given by $\prod_{i=1}^{k} \pi_{\sigma(i)}$, therefore the expected reward $R(S, \sigma)$ from selecting S and sequencing its jobs according to σ is

$$R(S, \sigma) = \sum_{k=1}^{|S|} r_{\sigma(k)} \prod_{i=1}^{k} \pi_{\sigma(i)}. \tag{1}$$

If we let $c(S) = \sum_{j \in S} c_j$ denote the total cost of selecting S, the *expected net profit* of selecting S and sequencing the jobs according to σ is

$$z(S, \sigma) = \sum_{k=1}^{|S|} r_{\sigma(k)} \prod_{i=1}^{k} \pi_{\sigma(i)} - c(S). \tag{2}$$

We study the problem of selecting a subset $S \subseteq J$ of jobs and finding a sequence σ so that the expected net profit $z(S, \sigma)$ is maximized. We call this problem the *Unreliable Job Selection and Sequencing Problem (UJSSP)*.

When there are no costs ($c_j = 0$ for all $j = 1, \ldots, n$), it is obviously profitable to select all jobs, and only the sequencing problem is left. In this case, an optimal sequence is obtained by simply scheduling the jobs by non-increasing values of the following index [Kadane, 1969, Mitten, 1960]:

$$Z_j = \frac{\pi_j r_j}{1 - \pi_j}. \tag{3}$$

Consequently, in UJSSP, once a subset S is selected, expression (2) is maximized by sequencing the jobs in S according to a schedule dictated by (3). Letting σ_Z denote such sequence, in the following we let $R(S) = R(S, \sigma_Z)$ and $z(S) = z(S, \sigma_Z)$. We can restate UJSSP as the problem of finding S^* such that:

$$z(S^*) = \max_{S \subseteq J} \{R(S) - c(S)\}.$$

For the sake of simplicity, unless specified otherwise, we assume from now on that the n jobs are numbered by non-increasing values of Z_j.

2 Complexity

Two special cases, namely with identical costs and with identical probabilities of success, can both be solved by a greedy algorithm. The greedy algorithm starts with the empty set and iteratively selects a job that yields the largest expected marginal gain in $z(S)$ until adding an extra job no longer leads to any improvement. This implies that both these special cases can be solved in $O(n^2)$. This result is deduced from the results from Stadje [1995] and Chade and Smith [2006] for the case of equal costs and from the results from Olszewski and Vohra [2016] for the case of equal probabilities.

In the general case, we prove that UJSSP is NP-hard by means of a reduction from the *Product Partition Problem (PPP)*, which is known to be strongly NP-hard [Ng et al., 2010]. For the proof, we refer to our working paper [Agnetis et al., 2025].

3 Exact solution methods

Exploiting the ordering induced by (3), we formulate a compact MILP formulation for UJSSP in which $x_j = 1$ if job j is selected, as follows:

$$\max \sum_{j=1}^{n} (r_j P_j - c_j x_j) \tag{4a}$$

$$P_j \leq \pi_j x_j \qquad \forall j \in \{1, \ldots, n\} \tag{4b}$$

$$P_j \leq \pi_j (P_i + 1 - x_i) \qquad \forall i, j \in \{1, \ldots, n\} : i < j \tag{4c}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \{1, \ldots, n\} \tag{4d}$$

$$P_j \geq 0 \qquad \forall\, j \in \{1, \ldots, n\} \tag{4e}$$

If $x_j = 1$, constraints (4b) and (4c) ensure that P_j is the product of the probabilities of the jobs selected up to j (included), whereas, if and only if job j is not selected, $P_j = 0$.

If all costs c_j are integer, UJSSP can also be solved using a backward dynamic programming (DP) algorithm. As this approach runs in pseudopolynomial time ($O(n \cdot \sum_{j \in J} c_j)$), we rule out strong NP-hardness for the integer-cost case. Let

the state be defined by a pair (b, i), where b denotes the remaining budget and i denotes that only jobs from the set $\{i, \ldots, n\}$ can be selected. The value function $g(b, i)$ represents the maximum attainable expected revenue under these constraints. The recurrence is given by

$$g(b,i) = \begin{cases} g(b, i+1) & \text{if } b \in \{0, \ldots, c_i - 1\}, \\ \max\left\{g(b, i+1),\ \pi_i \cdot \left(r_i + g(b - c_i, i+1)\right)\right\} & \text{if } b \in \{c_i, \ldots, \sum_{j \in J} c_j\}, \end{cases}$$

for $i < n$, while $g(b, n)$ is initialized with the value 0 for $b \in \{0, \ldots, c_n - 1\}$ and the value $\pi_n \cdot r_n$ for $b \in \{c_n, \ldots, \sum_{j \in J} c_j\}$. The optimal solution is obtained by evaluating $\max_b \{g(b, 1) - b\}$, which identifies the budget level that maximizes the objective value.

Our main contribution is two stepwise exact methods for solving UJSSP. The stepwise algorithms proceed stepwise through the jobs, forwards or backwards, and iteratively build and prune candidate subsets by exploiting structural properties of the objective function. For any set $S \subseteq J$ and any $j \in \{1, \ldots, n\}$, we define: $S_{<j} = S \cap \{1, \ldots, j-1\}$, $S_{\leq j} = S \cap \{1, \ldots, j\}$, $S_{>j} = S \cap \{j+1, \ldots, n\}$, and $S_{\geq j} = S \cap \{j, \ldots, n\}$. If $S^*_{>j}$ is given, then an optimal choice of earlier jobs $S^*_{\leq j} \subseteq J_{\leq j}$ must maximize:

$$R(S_{\leq j}) + \left(\prod_{i \in S_{\leq j}} \pi_i\right) \cdot R(S^*_{>j}) - c(S_{\leq j})$$

over all subsets $S_{\leq j} \subseteq J_{\leq j}$. If $S^*_{\leq j}$ is given, then an optimal continuation $S^*_{>j} \subseteq J_{>j}$ must maximize:

$$\left(\prod_{i \in S^*_{\leq j}} \pi_i\right) \cdot R(S_{>j}) - c(S_{>j})$$

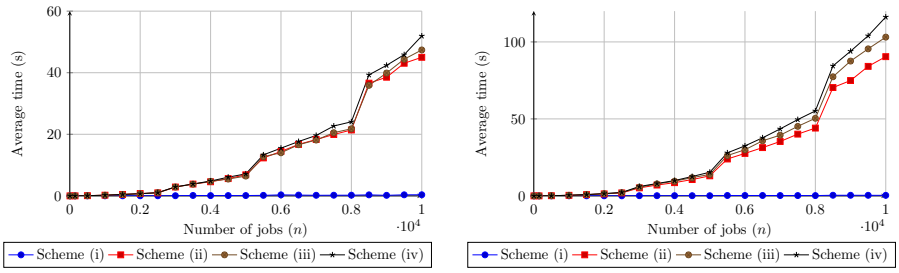
over all $S_{>j} \subseteq J_{>j}$. Our stepwise exact algorithms exploit the fact that, when S^* is unknown, both expressions depend linearly on a single unknown quantity that can be bounded: $R(S^*_{>j}) \in [0, R(J_{>j})]$ and $\prod_{i \in S^*_{\leq j}} \pi_i \in [\prod_{i \in J_{\leq j}} \pi_i, 1]$. Therefore, a candidate subset can only be part of an optimal solution if its corresponding linear function is part of the upper envelope of linear functions over the feasible range for the unknown quantity. If this is not the case, the subset is shown to be suboptimal and can be pruned (and so can any of its extensions). For more details on the stepwise algorithms we refer to our working paper [Agnetis et al., 2025].

4 Computational experiments

Using the MILP formulation, all instances with up to 50 jobs are solved to optimality within the 20-minute time limit, whereas no instance with more than 125 jobs is solved within this limit.

The DP algorithm is substantially more scalable: it solves all instances with up to 2,500 jobs within 10 minutes. However, instances with 3,000 jobs often take more than one hour of computation time.

The stepwise exact algorithms provide the strongest computational performance among all methods considered. The largest instances we generated ($n = 10,000$) were solved within one minute for the forward stepwise algorithm and within two minutes for the backward stepwise exact algorithm, as illustrated in Figure 1. For the stepwise algorithms, a second set of harder instances was constructed from the strongly NP-hard PPP. For these instances, solvability drops remarkably, instances with up to 20 integers are solved within 20 minutes, while none with more than 50 integers completed in that timeframe. By applying the stepwise methods to instances derived from the PPP, we provide, to the best of our knowledge, the first exact computational results reported for that problem.



(a) Forward method (b) Backward method

Fig. 1: Stepwise exact methods: average runtime vs. number of jobs

References

A. Agnetis, R. Leus, E. Perneel, and I. Salvadori. The unreliable job selection and sequencing problem, 2025. URL https://arxiv.org/abs/2511.17105.

H. Chade and L. Smith. Simultaneous search. *Econometrica*, 74(5):1293–1307, 2006.

J. Kadane. Quiz show problems. *Journal of Mathematical Analysis and Applications*, 27(3):609–623, 1969.

L. Mitten. An analytic solution to the least cost testing sequence problem. *Journal of Industrial Engineering*, 11(1):17, 1960.

C. T. Ng, M. Barketau, T. E. Cheng, and M. Y. Kovalyov. "product partition" and related problems of scheduling and systems reliability: Computational complexity and approximation. *European Journal of Operational Research*, 207(2):601–604, 2010.

W. Olszewski and R. Vohra. Simultaneous selection. *Discrete Applied Mathematics*, 200:161–169, 2016.

W. Stadje. Selecting jobs for scheduling on a machine subject to failure. *Discrete Applied Mathematics*, 63(3):257–265, 1995.

Branch-and-Price for Job-Shop Scheduling with Time-Dependent Costs and Cardinality Constraints

Marouane Felloussi[1,2], Mohammed Ghannam[2], João Dionísio[2,3], Paolo Gianessi[1], Xavier Delorme[1]

[1] Mines Saint-Etienne, CNRS, UMR 6158 LIMOS, Saint-Etienne, France
[2] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
[3] Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre s/n, Porto, Portugal
{felloussi,ghannam,dionisio}@zib.de; {delorme,paolo.gianessi}@emse.fr

Keywords: Branch and price, Job-shop scheduling, Integer programming.

1 Introduction

Time-dependent operating costs and resource limits arise naturally in shop-floor scheduling, driven by fluctuating operating rates such as energy prices and contractual peak-capacity constraints. In these settings, schedules are evaluated not only by traditional metrics such as makespan, but also by how resource usage is allocated over time. This motivates scheduling models with time-dependent processing costs and capacity limits.

Time-indexed formulations not only represent a natural modeling choice, but also typically yield strong linear relaxations (Artigues 2017). Their main drawback, however, is size: they scale poorly with the horizon length, leading to pseudo-polynomial models that are often difficult to solve directly. Period-indexed variants (Felloussi *et al.* 2025) apply when costs are piecewise constant (e.g., Time-of-Use pricing); hence, they can reduce model size as the number of intervals is orders of magnitude smaller than the number of time steps. However, they do not address settings where costs and limits vary at unit-time resolution. Decomposition-based extended formulations have proven effective in parallel-machine environments (Pessoa *et al.* 2010), but their applications to shop scheduling remain limited and computationally challenging (Lancia *et al.* 2007). Moreover, this problem class can exhibit several practical difficulties documented for branch and price (Kolter *et al.* 2025), notably master problem degeneracy and dual oscillations.

We study a job-shop scheduling variant with time-dependent costs and cardinality capacity constraints and propose a reformulation embedded in a branch-and-price framework. Our core design goal is to preserve the strong dual bounds of disaggregated time-indexed formulations while shifting some computational effort to a fast integer pricing oracle. The branching rules are enforced in the integer subproblem, and the variable selection incorporates elements from the problem structure together with a hybrid of strong and pseudo-cost branching. This is complemented by local propagation rules tailored to the disjunctive and partitioning structure of the problem. We further exploit the framework and its underlying structure through fast primal heuristics. In addition, we incorporate enhancements such as early branching and dual smoothing. We evaluate the resulting framework through computational experiments on a large and diverse set of job-shop benchmark instances.

2 Branch-and-Price approach

We consider a set of machines $\mathcal{M}$ and a set of jobs $\mathcal{J}$ over a discrete time horizon $\mathcal{T}$. Each job $j \in \mathcal{J}$ consists of an ordered sequence of $|\mathcal{M}|$ operations, subject to precedence constraints. We write $(j,m) \prec (j,m')$ if m precedes m' in the sequence of job j. Operation (j,m) has processing time $q_{j,m}$. At time step $t \in \mathcal{T}$, a cardinality constraint limits the number of active machines to at most $\overline{M}_t$. Executing (j,m) at time t incurs a cost $g^t_{j,m}$.

Extended formulation. Time-dependent costs and cardinality constraints call for a time-indexed formulation. A Dantzig-Wolfe reformulation decomposes the problem into job-wise schedules, yielding an extended formulation with exponentially many variables, whose relaxation is solved by column generation. For each job $j \in \mathcal{J}$, let $\mathcal{P}_j$ be the set of feasible job-schedules, and $\mathbf{P} \coloneqq \bigcup_{j\in\mathcal{J}} \mathcal{P}_j$. Each column $p \in \mathcal{P}_j$ is represented by a profile $a^p_{m,t} \in \{0,1\}$ and has cost $c_p \coloneqq \sum_{m\in\mathcal{M}} \sum_{t\in\mathcal{T}} g^t_{j,m} a^p_{m,t}$. The master problem (**MP**) reads

$$
\begin{aligned}
\textbf{(MP)} \quad \min \quad & \sum_{p\in\mathbf{P}} c_p \lambda_p, & & \\
\text{s.t.} \quad & \sum_{p\in\mathcal{P}_j} \lambda_p = 1, & \forall j \in \mathcal{J} \quad & [\alpha_j] \\
& \sum_{p\in\mathbf{P}} a^p_{m,t} \lambda_p \leqslant 1, & \forall m \in \mathcal{M}, \forall t \in \mathcal{T} \quad & [\beta_{m,t}] \\
& \sum_{p\in\mathbf{P}} \sum_{m\in\mathcal{M}} a^p_{m,t} \lambda_p \leqslant \overline{M}_t, & \forall t \in \mathcal{T} \quad & [\gamma_t] \\
& \lambda_p \geqslant 0, & \forall p \in \mathbf{P}. &
\end{aligned}
\tag{1}
$$

Pricing subproblem. The integer pricing subproblem either finds a column with negative reduced cost for the restricted master problem (**RMP**) or certifies dual feasibility. Let $\hat{\alpha}_j$, $\hat{\beta}_{m,t}$, and $\hat{\gamma}_t$ be optimal dual variables after solving (**RMP**) and define $\hat{c}^t_{j,m} \coloneqq g^t_{j,m} - \hat{\beta}_{m,t} - \hat{\gamma}_t$. The reduced cost of $p \in \mathcal{P}_j$ is $-\hat{\alpha}_j + \sum_{m,t} \hat{c}^t_{j,m} a^p_{m,t}$, and pricing for $j \in \mathcal{J}$ reduces to solving $\min_{p\in\mathcal{P}_j} \sum_{m,t} \hat{c}^t_{j,m} a^p_{m,t}$. A feasible partial schedule specifies, for each operation (j,m) of index i, a completion time t at its dedicated machine m respecting 1) operation order, 2) processing of i on m for $q_{j,m}$ consecutive time steps, and 3) execution of i starts only after completion of $i-1$. In the following, we reason about a fixed job j. For convenience, we index its operations by their position i, and write $q_i \equiv q_{j,m}$ and $\hat{c}^\tau_{j,m} \equiv \hat{c}^\tau_i$. Finding a minimum cost partial schedule can be described as a dynamic program (DP) on the state (i,t). Let $d^\star_j(i,t)$ denote the minimum cost of any partial schedule completing operation i at time t, with $d^\star_j(0,t) = 0$ and infeasible states set to $+\infty$. Let $S^t_i \coloneqq \sum_{\tau=t-q_i+1}^{t} \hat{c}^\tau_i$. Each state follows from either an idle or a processing transition, and the recurrence is

$$
d^\star_j(i,t) = \min\Big\{ d^\star_j(i,t-1), d^\star_j(i-1,t-q_i) + S^t_i \Big\}, \tag{2}
$$

which is nonincreasing in t. Hence, the minimum reduced cost of any complete pattern of j equals $d^\star_j(|\mathcal{M}|,|\mathcal{T}|)$. Equivalently, this is a shortest-path problem on a directed acyclic graph whose node pairs are (i,t), and whose arcs represent idle transitions of zero length and processing transitions of length S^t_i. The convex hull of these paths is integral. Therefore, this reformulation preserves the LP bound of a disaggregated compact formulation, while reducing the effective problem size through decomposition.

Branching. Column generation is embedded in an exact branch-and-price scheme to solve the integer problem. To obtain convexified branching constraints, we branch on original variables $z^t_{j,m}$, denoting whether operation (j,m) is processed at time step t. The two branches $z^t_{j,m} = 1$ and $z^t_{j,m} = 0$ correspond to forcing or forbidding a set of transitions in the integer subproblem. The branching variable selection follows a *most-dispersion rule*. In any integer feasible solution, operations execute over a contiguous time block; the branching rule exploits this and chooses the operation whose execution is most dispersed in a fractional solution and the most-used time step. Precisely, we define $F_{j,m} \coloneqq \{t \in \mathcal{T} : 0 < z^{\star t}_{j,m} < 1\}$ and choose $(j,m) = \arg\max_{(j',m')} |F_{j',m'}|$; then, we take $t = \arg\max_{t\in F_{j,m}} z^{\star t}_{j,m}$.

3 Acceleration techniques

Reliable pseudo-cost variable selection. In MIP, a standard variable selection strategy is pseudo-cost branching (Benichou *et al.* 1971), which provides a problem-agnostic and instance-adaptive ranking of branching candidates. Since pseudo-costs are initially unavailable, the most-dispersed rule is first used to filter candidates. On this reduced set, strong branching without pricing is performed, and the variable with the best combined bound improvement is selected (Achterberg 2007). This hierarchical strong branching scheme is applied only near the root. A reliability threshold determines when pseudo-costs are deemed usable with additional strong branching executed within a fixed budget, with a most-dispersed rule fallback. These pseudo-cost scores are complemented with aggregated pseudo-cost information computed over variable sets: average dual gains are maintained by operation (j, m) and per time step t, and combined with the variable-level pseudo-cost through a mixing parameter. This idea exploits that branching on $z^t_{j,m}$ also restricts decisions related to operation (j, m) and to time step t.

Local propagation. Let $\mathcal{B}_n$ denote the set of branching constraints on the path to node n. To ensure consistency with the branching decisions, we fix to zero any column that violates $\mathcal{B}_n$. Furthermore, we derive the following fixings from the problem structure:

Job and machine disjunction: If $\{z^t_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j,m'} = 0\}$ for all $m' \neq m$; $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j',m} = 0\}$ for all $j' \neq j$.

Precedence: If $\{z^t_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^{t'}_{j,m'} = 0\}$ for $(j, m') \prec (j, m)$ and $t' > t$. A similar argument holds for succeeding operations.

Non-preemption (contiguity): If $\{z^{t_1}_{j,m} = 1\}, \{z^{t_2}_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j,m} = 1\}$ for $t_1 < t < t_2$.

Non-preemption (interruption): If $\{z^{t_1}_{j,m} = 0\}, \{z^{t_2}_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j,m} = 0\}$ for $t < t_1$ if $t_1 < t_2$, and for $t > t_2$ if $t_2 < t_1$.

Active machines limit: For $t \in \mathcal{T}$ and $\mathcal{M}' \subset \mathcal{M}$ with $|\mathcal{M}'| = \overline{M}_t$, if $\forall m \in \mathcal{M}'$, $\exists j \in \mathcal{J}, \{z^t_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j,m'} = 0\}$ for all $m' \in \mathcal{M} \setminus \mathcal{M}'$ and $j \in \mathcal{J}$.

This propagation routine is also invoked during strong branching tests. Furthermore, the number of implications produced by a candidate branching is used as an inference score, as in hybrid pseudo-cost inference branching (Achterberg 2007).

Primal heuristics. At each B&P node, primal solutions must be constructed from the current column pool. Generic MIP heuristics apply but are agnostic to the column generation process, and cannot distinguish heuristic failure from the absence of a feasible solution in the 0–1 (**RMP**). We therefore design a dedicated branch-and-bound scheme with problem-specific bounding and a large-neighborhood search, outlined below.

1) Exact tree search. The restricted master heuristic solves the 0–1 (**RMP**) induced by the column pool available at the current node, via a depth-first branch-and-bound that exploits propagation from the set-partitioning, clique, and resource constraints. Jobs are processed in decreasing order of available columns, and each job's columns are tried in non-decreasing cost order. Along a DFS path, the state is given by the set of occupied machine-time pairs and the per-time usage counts; selecting a column updates these, and any disjunction or time-capacity violation yields immediate pruning. A node bound is computed by adding, for each remaining job, the cheapest column compatible with the current state, and the node is pruned whenever this bound is no better than the incumbent.

2) Large Neighborhood Search: The LNS runs destroy–repair cycles on integer solutions found during branch and price, or on partial solutions from the exact tree search. Given an incumbent λ^{inc} and a destroy set $D \subset \mathcal{J}$, the neighborhood keeps $\lambda_p = \lambda^{\text{inc}}_p$ for all $j \notin D$ and reoptimizes the remaining jobs over the same clique and time-capacity constraints.

This is embedded in an iterative destroy–repair loop; the repair step relies on the same DP oracle as pricing with a modified objective and the fixings induced by the neighborhood.

4 Computational experiments and conclusion

We evaluate the framework using 448 instances generated following the scheme of Masmoudi *et al.* (2019). The set is obtained by combining seven base job-shop instances with $|\mathcal{J}| \in \{5, 6, 7, 8\}$ and $|\mathcal{M}| \in \{6, 7, 8, 10\}$, and varying operation duration ranges. They are extended with operational costs of the form $g^t_{j,m} = \varphi_m \kappa^t$, where φ_m models machine-dependent intensity and κ^t is a time-dependent price profile. We consider four value sets for φ inspired by knapsack (KP) sets (Hojny *et al.* 2020): small- and large-coefficient, three-coefficient, arithmetic-sequence, and weakly super-increasing KP sets. For κ, we use two time-of-use profiles inspired by electricity tariff structures (Felloussi *et al.* 2025), and two uniformly random profiles. The maximum number of active machines is fixed for all $t \in \mathcal{T}$ to values in $\{|\mathcal{M}| - 1, |\mathcal{M}| - 2\}$. Planning horizons are set as $|\mathcal{T}| = \lceil \omega C^\star \rceil$, with expansion factor $\omega \in \{1.2, 1.5\}$, where $C^\star$ is the makespan of the base instance under the cardinality constraint. The values of $C^\star$ range from 56 to around 1000 for the largest instances.

Experiments were executed single-threaded on an Intel Xeon Gold 5122 CPU with 96GB of RAM and a one-hour time limit. We used the Python interface of SCIP (Hojny *et al.* 2025, Maher *et al.* 2016) as the branch-and-price framework to gain fine-grained control over the solving process, with SoPlex as the LP-solver. The shortest-path pricing oracle was implemented in C, while all other components were implemented in Python. Since pricing is solved exactly, we compute Lagrangian bounds (Lübbecke and Desrosiers 2005) and apply integrality-based early branching (Mehrotra and Trick 1996), as well as the adaptive dual smoothing scheme of Pessoa *et al.* (2018). Averages over instance subsets are computed using the shifted geometric mean with shift 1. We partition the instances by time horizon length and compare the proposed branch-and-price framework to an aggregated time-indexed (TI) formulation (Artigues 2017). We omit results for the disaggregated formulation, as it failed to solve the root LP within the time limit on medium instances and exceeded memory limits on larger instances.

Table 1: Performance comparison of the aggregated compact formulation and the proposed branch-and-price framework.

Instances		Compact			Branch-and-Price			Relative	
subset	#inst.	#solved	time (s)	#nodes	#solved	time (s)	#nodes	solved	time
small	128	126	275.72	130.15	128	22.83	23.51	1.02	0.08
medium	128	19	3339.13	209.42	104	245.74	18.57	5.47	0.07
large	192	1	3593.45	10.98	107	1094.14	14.55	107	0.30
all	448	146	1342.25	49.02	**339**	**175.42**	**15.52**	2.32	0.13

Table 1 reports the computational results. The extended formulation solved by branch and price (B&P) proves consistently more reliable than the aggregated time-indexed (TI) formulation, with the performance gap increasing as the time horizon grows. This difference appears both in the time to optimality and in the number of instances solved to completion. This advantage reflects both the (theoretically) tighter relaxation provided by the extended formulation and an algorithmic structure that shifts part of the horizon-dependent effort to pricing. In contrast, the aggregated TI formulation yields slower LP relaxations, which inflate per-node costs and increasingly dominate the computation as the horizon expands, as reflected by the runtime and node counts. Within B&P, a substantial portion of this

horizon-driven combinatorial burden is absorbed by the fast pricing oracle, while structure-aware mechanisms lead to smaller search trees.

Beyond the empirical improvements, we observe that the integrality property of the pricing subproblem need not be a limitation in this time-indexed setting. The LP relaxation of the disaggregated TI formulation is already strong, and the main benefit of decomposition lies in the computational leverage provided by a fast pricing oracle. This efficiency enables components that rely on repeated oracle calls, like destroy–repair heuristics and dual stabilization. The framework accommodates further enhancements. One direction is the integration of combinatorial strengthening cuts and refined branching schemes that incorporate problem-specific structure with standard MIP techniques to mitigate degeneracy. The availability of a fast oracle further enables other procedures requiring multiple pricing evaluations, such as subgradient-based updates. Finally, the approach extends naturally to variants with alternative machine assignments per operation, including flexible job-shop settings, with only moderate adaptations to pricing and the associated components.

References

Achterberg, T., 2007. Constraint integer programming. Doctoral dissertation, Technische Universität Berlin.

Van den Akker, J.M., Hurkens, C.A. and Savelsbergh, M.W., 2000. Time-indexed formulations for machine scheduling problems: Column generation. INFORMS Journal on Computing, 12(2), pp.111-124.

Artigues, C., 2017. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. Operations Research Letters, 45(2), pp.154-159.

Benichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribiere, G. and Vincent, O., 1971. Experiments in mixed-integer linear programming. Mathematical Programming, 1(1), pp.76-94.

Felloussi, M., Delorme, X. and Gianessi, P., 2025, February. Minimizing Energy Cost in a Job-Shop Scheduling Problem Under ToU Pricing: A New Method Based on a Period-Indexed MILP. In 14th International Conference on Operations Research and Enterprise Systems (pp. 320-327).

Hojny, C., Gally, T., Habeck, O., LÃ¼then, H., Matter, F., Pfetsch, M.E. and Schmitt, A., 2020. Knapsack polytopes: a survey. Annals of Operations Research, 292(1), pp.469-517.

Hojny, C., Besancon, M., Bestuzheva, K., Borst, S., Dionisio, J., Ehls, J., Eifler, L., Ghannam, M., Gleixner, A., Goss, A. and Hoen, A., 2025. The SCIP Optimization Suite 10.0. arXiv preprint arXiv:2511.18580.

Kolter, M., Grunow, M. and Kolisch, R., 2025. On Branch-and-Price for Project Scheduling. arXiv preprint arXiv:2501.04563.

Lancia, G., Rinaldi, F. and Serafini, P., 2007. A compact optimization approach for job-shop problems. In Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: theory and applications (MISTA) (pp. 293-300).

Lübbecke, M.E. and Desrosiers, J., 2005. Selected topics in column generation. Operations research, 53(6), pp.1007-1023.

Maher, S., Miltenberger, M., Pedroso, J.P., Rehfeldt, D., Schwarz, R. and Serrano, F., 2016, July. PySCIPOpt: Mathematical programming in Python with the SCIP optimization suite. In International Congress on Mathematical Software (pp. 301-307). Cham: Springer International Publishing.

Masmoudi, O., Delorme, X. and Gianessi, P., 2019. Job-shop scheduling problem with energy consideration. International Journal of Production Economics, 216, pp.12-22.

Mehrotra, A. and Trick, M.A., 1996. A column generation approach for graph coloring. INFORMS Journal on Computing, 8(4), pp.344-354.

Pessoa, A., Uchoa, E., De Aragão, M.P. and Rodrigues, R., 2010. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. Mathematical Programming Computation, 2(3), pp.259-290.

Pessoa, A., Sadykov, R., Uchoa, E. and Vanderbeck, F., 2018. Automation and combination of linear-programming based stabilization techniques in column generation. INFORMS Journal on Computing, 30(2), pp.339-360.

Mixed-Integer Programming for the Resource Investment Problem with Gap-Free Processing Requirements

Jonas Saupe[1], Jasmin Montalbano[1], and Stefan Nickel[1 2]

[1] FZI Research Center for Information Technology, Karlsruhe, Germany
{saupe, montalbano, nickel}@fzi.de
[2] Karlsruhe Institute of Technology, Germany
stefan.nickel@kit.edu

Keywords: Mixed-Integer Linear Programming, Resource-Constrained Project Scheduling, Resource Management.

1 Introduction

Carrying out projects requires various kinds of resources, such as staff or machines. Sufficient resource capacities have to be acquired whenever a project has to be completed within a certain time. The resource investment problem (RIP) is about determining a project schedule that minimizes the total cost of required capacities subject to precedence constraints. Extending the problem by minimum and maximum time lags between certain jobs yields the RIP with generalized temporal constraints (RIP/max). In this paper, we consider a novel variant of RIP/max containing *gap-free processing (GF) requirements* on some disjoint subsets of jobs. For each subset, at least one of its jobs must be in process during each period from the earliest start- to the latest completion period of any job in the set. GF is motivated by an application in infrastructure maintenance, where it ensures a uniform technical specification of facilities within a region. Specifically, maintenance of certain equipment, such as telecommunication technology, is scheduled without gaps across all facilities managed by the same control center. We refer to this extension as the *RIP with generalized temporal constraints and gap-free processing requirements (RIP/max-gf)*.

GF does not prescribe a predefined execution order or sequential processing, unlike no-wait constraints [Allahverdi, 2016]. Also, unlike contiguity in the context of batch scheduling [Potts and Kovalyov, 2000], it does not permit periods in which no job from a set is in process. We are aware of the state-of-the-art approaches for the RIP [Rodrigues and Yamashita, 2014,Kreter et al., 2018]. However, we propose a mixed-integer programming (MIP) model as a baseline method for RIP/max-gf. Our contributions are a precise problem description, an MIP formulation, and computational experiments illustrating that GF requirements impose a substantial computational burden in an MIP-based approach.

The remainder of this paper is organized as follows. We formally define the RIP/max-gf in Section 2 and describe the MIP model in Section 3. Section 4 summarizes our computational results. Section 5 provides a conclusive outlook on future work.

2 Problem statement

We are given a set $\mathcal{J}$ of jobs, a set $\mathcal{R}$ of resources and a set $\mathcal{T}$ of periods. Let c_k denote the cost per capacity unit of resource $k \in \mathcal{R}$. We have to select a start period $S_j \in \mathcal{T}$ for each job $j \in \mathcal{J}$ and a capacity $R_k \geq 0$ for each resource $k \in \mathcal{R}$ such that the total capacity cost $\sum_{k \in \mathcal{R}} c_k R_k$ is minimized. Four sets of constraints apply. First, *non-preemption constraints* impose that each job starts exactly once. Second, each job $j \in \mathcal{J}$ has a duration of p_j periods and requires r_{jk} units of capacity from resource $k \in \mathcal{R}$ during each period of processing. *Capacity constraints* require that for each period $t \in \mathcal{T}$ and each resource $k \in \mathcal{R}$, the total capacity requirement of jobs that are in process during period t does not exceed the chosen capacity R_k. Third, *generalized temporal constraints* on certain pairs of jobs $(i, j) \in E \subseteq \mathcal{J} \times \mathcal{J}$ impose a minimum start-to-start time lag of δ_{ij}, i.e., $S_i + \delta_{ij} \leq S_j$ for all $(i, j) \in E$. Lastly, *GF requirements* are specified by a collection $\mathcal{G}$ of disjoint *gap-free sets* $G \subseteq \mathcal{J}$. For such a set $G \in \mathcal{G}$, ES_G (LC_G) denotes the earliest start period (latest completion period) of any job $j \in G$. GF prescribes that for all $G \in \mathcal{G}$ and all $t \in \{ES_G, \ldots, LC_G\}$, at least one job $j \in G$ must be in process during t, i.e., $S_j \leq t < S_j + p_j$.

3 Mixed-integer programming formulation

We model RIP/max-gf on a flow network $(\mathcal{V}, \mathcal{E})$ with a set $\mathcal{V}$ of vertices corresponding to job-start period combinations $(j, t) \in \mathcal{J} \times \mathcal{T}$. We assign a flow $f_e \in \{0, 1\}$ to each edge $e = ((j, t), (j', t')) \in \mathcal{E}$. $f_e = 1$ models that j is scheduled in period t and j' in period t'. For each set $G \in \mathcal{G}$ and distinct $j, j' \in G$, $\mathcal{E}$ contains an *interior edge* $((j, t), (j', t'))$ if $t' \in \{t, \ldots, t + p_j\}$, i.e., if $S_j = t$ and $S'_j = t'$ does not contradict GF. Furthermore, we introduce a dummy source α and a dummy sink ω adjacent to all non-dummy vertices. For any $G \in \mathcal{G}$, we refer to the set of edges from α to any vertex (j, t) $(j \in G, t \in \mathcal{T})$ as its *start edges*. $\mathcal{E}_v^{\text{in}}$ ($\mathcal{E}_v^{\text{out}}$) denotes the set of ingoing (outgoing) edges of vertex $v \in \mathcal{V}$.

We can express several constraints of RIP/max-gf in terms of *subsets of edges with a minimum or maximum total flow*. Let $\mathcal{M}^{\min}$ ($\mathcal{M}^{\max}$) be the set of tuples (M, c), where c is the minimum (maximum) total flow capacity that applies to $M \subseteq \mathcal{E}$. Then, the non-preemption constraint for job $j \in \mathcal{J}$ is expressed by a tuple $(M, 1)$ added to both $\mathcal{M}^{\min}$ and $\mathcal{M}^{\max}$, where M contains all outgoing edges of vertices (j, t) $(t \in \mathcal{T})$. A temporal constraint for $(i, j) \in E$ is modeled by adding a tuple $(M, 1)$ to $\mathcal{M}^{\max}$ for each period $t \in \mathcal{T}$, where M contains all outgoing edges of vertices (i, τ) with $\tau \geq t$ and all outgoing edges of vertices (j, τ') with $\tau' < t + \delta_{ij}$. The GF requirement for $G \in \mathcal{G}$ is enforced by adding a tuple $(M, 1)$ to $\mathcal{M}^{\max}$, where M is the set of start edges of G. As a result, only one job in G receives flow directly from the source α and non-preemption constraints for jobs $j' \in G \setminus \{j\}$ have to be satisfied by assigning flow to interior edges. By the definition of the edge set and flow conservation, this ensures that the start periods of jobs in G satisfy GF. Finally, for $t \in \mathcal{T}$, let $\mathcal{A}_t \subseteq \mathcal{E}$ denote the set of edges $e = ((j, \tau), (j', \tau'))$ for which $\tau \leq t < \tau + p_j$, i.e., the set of

edges contributing a capacity requirement of $\tilde{r}_{ek} \coloneqq r_{jk}$ in period t. Altogether, we can state the MIP formulation for the RIP/max-gf as follows:

$$\min \sum_{k\in\mathcal{R}} c_k R_k \tag{1}$$
$$\text{s.t.} \sum_{e\in\mathcal{E}_v^{\text{in}}} f_e = \sum_{e\in\mathcal{E}_v^{\text{out}}} f_e, \qquad v \in \mathcal{V} \tag{2}$$
$$\sum_{e\in\mathcal{A}_t} \tilde{r}_{ek} f_e - R_k \leq 0, \qquad t \in \mathcal{T}, k \in \mathcal{R} \tag{3}$$
$$\sum_{e\in M} f_e \geq c, \qquad (M,c) \in \mathcal{M}^{\min} \tag{4}$$
$$\sum_{e\in M} f_e \leq c, \qquad (M,c) \in \mathcal{M}^{\max} \tag{5}$$
$$f_e \in \{0,1\}, \qquad e \in \mathcal{E}$$

(1) minimizes the total capacity cost. (2) are flow conservation constraints. Capacity constraints (3) are modeled in the standard manner based on the sets $\mathcal{A}_t$. (4) and (5), respectively, enforce minimum and maximum total flows.

The example in Figure 1 shows the network for an instance with four time periods and $\mathcal{G} = \{\{1,2\}\}$. Jobs 1 and 2 have durations of $p_1 = 1$ and $p_2 = 2$. The total flow on the start edges of $\{1,2\}$ (dotted in Figure 1) may be at most 1. Thus, some interior edges (dashed in Figure 1) must be assigned positive flow to satisfy all non-preemption constraints. This prevents infeasible schedules, since, for instance, no edge $((1,1),(2,3))$ exists.

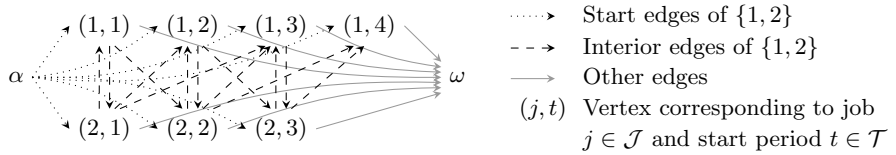


Fig. 1. Network for a two-job example instance ($\mathcal{G} = \{\{1,2\}\}$)

4 Computational results

Our computational evaluation aims to study the effect of gap-free processing requirements on solution times and quality. For this purpose, we enriched 210 benchmark RIP/max instances containing 30 jobs each [Kolisch et al., 1999] with random gap-free sets of varying size and number. We compare our MIP model to a standard discrete-time formulation for the RIP/max. Experiments ran on an AMD Ryzen 7 2.70 GHz machine with 32 GB RAM; models were solved by Gurobi 10.0.3 within a 60 seconds time limit.

The results of our computational evaluation are summarized below. Instances in tables 1 and 2 are grouped by $|\mathcal{G}|$. For each group, the number of instances is shown in column #. Table 1 compares the average solution times for instances solved to optimality within the time limit. Table 2 compares the average optimality gaps for instances that were not solved to optimality within the time limit. Gaps are computed as $(UB - LB)/UB$, where UB and LB, respectively, denote the best upper- and lower bound found by the solver. The columns corresponding to RIP/max only contain results for the base instances without added gap-free sets ($|\mathcal{G}| = 0$), as gap-free sets are not represented in this model.

$\lvert\mathcal{G}\rvert$	#	Avg. time (ms)	
		RIP/max-gf	RIP/max
0	18	6638	65
$\lceil 0.1\lvert\mathcal{J}\rvert\rceil$	44	14291	
$\lceil 0.3\lvert\mathcal{J}\rvert\rceil$	34	16004	
$\lceil 0.5\lvert\mathcal{J}\rvert\rceil$	37	17386	
	133	14554	65

Table 1. Average solution times for instances for which the RIP/max-gf model was solved to optimality

$\lvert\mathcal{G}\rvert$	#	Avg. gap (%)	
		RIP/max-gf	RIP/max
0	3	3.43	0
$\lceil 0.1\lvert\mathcal{J}\rvert\rceil$	19	6.97	
$\lceil 0.3\lvert\mathcal{J}\rvert\rceil$	29	9.34	
$\lceil 0.5\lvert\mathcal{J}\rvert\rceil$	26	6.97	
	77	7.72	0

Table 2. Average optimality gaps for instances for which the RIP/max-gf model was not solved to optimality

First, we observe that GF is indeed meaningful, as it increases the optimal cost by 61% on average. Second, the RIP/max-gf model contains about 100 times more variables and about ten times more constraints than the RIP/max model, which explains the vast increase in solution times shown in Table 1. Even for instances with $|\mathcal{G}| = 0$, solution times are about 100 times longer for the RIP/max-gf model. Increasing $|\mathcal{G}|$ seems to further complicate the problem. This matches the results in Table 2, which show consistently larger optimality gaps when $\mathcal{G} \neq \emptyset$, despite the lack of a clear correlation with $|\mathcal{G}|$.

5 Conclusions and future work

In this paper, we described a novel generalization of the RIP/max consisting of the introduction of gap-free processing requirements and proposed an MIP formulation. Our computational analysis demonstrates that the extensive modeling required to capture gap-free processing in an MIP-based approach leads to significantly higher computation times compared to a standard formulation. This emphasizes the need for future research on more sophisticated solution techniques for the RIP/max-gf, such as, e.g., decomposition and heuristics.

References

[Allahverdi, 2016] Allahverdi, A. (2016). A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255(3):665–686.

[Kolisch et al., 1999] Kolisch, R., Schwindt, C., and Sprecher, A. (1999). *Benchmark Instances for Project Scheduling Problems*, pages 197–212. Kluwer, Boston.

[Kreter et al., 2018] Kreter, S., Schutt, A., Stuckey, P. J., and Zimmermann, J. (2018). Mixed-integer linear programming and constraint programming formulations for solving resource availability cost problems. *European Journal of Operational Research*, 266(2):472–486.

[Potts and Kovalyov, 2000] Potts, C. N. and Kovalyov, M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249.

[Rodrigues and Yamashita, 2014] Rodrigues, S. B. and Yamashita, D. S. (2014). *Exact Methods for the Resource Availability Cost Problem*, pages 319–338. Springer International Publishing.

D - Heuristics and metaheuristics #1

A novel scheduling technique for NPV improvement of resource-constrained project scheduling problem with discounted cash flows

Yuqin An[1], Mario Vanhoucke[1,2,3]

[1] Faculty of Economics and Business Administration, Ghent University,
Yuqin.An@Ugent.be, Mario.Vanhoucke@Ugent.be
[2] Operations and Technology Management Centre, Vlerick Business School, Belgium
[3] UCL School of Management, University College London, United Kingdom

Keywords: resource constraint, discounted cash flows, NPV improvement, scheduling technique.

1 Introduction

As a standard extension of the resource constrained project scheduling problem (RCPSP), the RCPSP with discounted cash flows (RCPSPDC) shifts the optimization focus from time to economic value by maximizing the project's net present value (NPV) within a given deadline. Given its explicit focus on financial performance, which is an objective of central importance in practical project management, the RCPSPDC has attracted extensive attention in the literature [Artigues et al., 2025]. Early research focused on developing exact solution procedures, and in parallel heuristic approaches were proposed to provide more intuitive strategies. However, due to the NP-hard nature of RCPSPDC and the computational effort required by exact algorithms, the focus gradually shifted toward metaheuristic techniques, which remains the dominant research direction. In most research, metaheuristic frameworks were supplemented with classical local search mechanisms to enhance solution quality. Also, activity-move rules inspired by intuitive heuristics, such as advancing cash inflows and delaying cash outflows, were incorporated to further improve NPV. In contrast, Leyman and Vanhoucke [2015] introduced a scheduling technique specifically designed to enhance NPV, and it was embedded into the metaheuristic as a local search method. To the best of our knowledge, this represents the most recent scheduling technique proposed for NPV improvement. Nevertheless, this technique either only delays or advances activities according to the cash flow (CF) distribution in a project, overlooking additional improvement opportunities arising from bidirectional modifications.

Building on these grounds, this study proposes a forward-backward scheduling technique that simultaneously incorporates both forward and backward adjustments, rather than relying on a single directional shift. Furthermore, the technique integrates local rescheduling with different strategies during adjustments, which has the potential to expand effective search space.

2 Methodology

The forward-backward scheduling technique consists of 3 consecutive steps, shown in Fig.1. Step 1 is the essential step, aiming to generate high-quality initial solution. Based on the initial solution, Step 2 aims to improve NPV by advancing activities with positive CFs, during which local rescheduling is performed, leading to the forward-moved schedule. Then, based on the forward-moved schedule, Step 3 is performed to improve NPV further by delaying activities with negative CFs, while simultaneously conducting local rescheduling, resulting in the final optimized schedule. Details of each step are illustrated below.

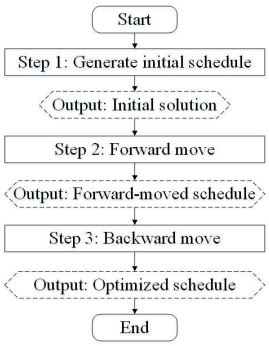


Fig. 1. Workflow of the forward-backward scheduling technique

Step 1: generate initial schedule

To create high-quality initial solution, we adopt the method proposed by Leyman and Vanhoucke [2015].

Step 2: forward move (FM)

The FM performs with the forward activity list to advance activities as early as possible according to the 4 procedures below. The list is generated according to the start time of each activity in the initial schedule in ascending order. Then for each activity i in order:

1. **Check CF.** Only if the activity has positive CF, the FM proceeds, any activity with negative CF will be skipped.
2. **Determine the maximum allowable forward shift** α_F. This means the maximum allowable advance of activity i. $\alpha_F = \max(t_{ext}, s_i - f_j)$, where t_{ext} is the time point corresponding to the NPV extremum of activity i, in this step $t_{ext} = 0$, s_i is the start time of activity i, f_j is the finish time of the latest-finishing predecessor j of activity i in current schedule.
3. **Execute the FM actions.** For a given α_F, two consecutive move actions are proposed to obtain a candidate schedule. First, move activity i forward directly by up to α_F time units in current schedule. In this way, the current schedule is changed, leaving resources idle at previous position of activity i

and possibly inducing resource conflicts at its new position. Therefore, the second move action, a local rescheduling is proposed. In specific, all activities are divided into two sets at first according to their relationships with activity i, i.e. the fixed activity set (FAS) and the reschedulable activity set (RAS). Then, the positions of all activities $i^{'} \in FAS$ are kept unchanged in the current schedule. Subsequently, all activities $i^{''} \in RAS$ are rescheduled using two different strategies based on current schedule. Strategy 1 reschedules eligible activities using the same priority list as the initial schedule. Strategy 2 differs from Strategy 1 in that it reschedules activities using a different priority list from the initial schedule, which expands search space further. After these two move actions, a candidate schedule can be obtained, which will be checked in procedure 4.

4. **Check the candidate schedule and output the forward-moved schedule**. With the aim of maximizing NPV, not only is the deadline constraint of the candidate schedule checked, but a greedy strategy is also applied to determine whether the NPV is improved. When the candidate schedule is deadline-feasible and its NPV is improved relative to current schedule, the current schedule is updated with it, and same procedures above are performed on the next activity based on it. Otherwise, the new technique reduces α_F step by step and performs FM actions on activity i until a deadline-feasible, NPV-improved candidate schedule is achieved. When α_F is reduced to 0 and no accepted candidate schedule is found, current schedule is retained and still used as the basis for proceeding to the next activity. Notably, for the FM of the first activity in the activity list, the initial schedule serves as the current schedule. However, for all of the subsequent activities, the basis for their FMs, i.e. current schedule, is iteratively updated with the deadline-feasible, NPV-improved candidate schedule when available. After traversing all activities in the activity list, the latest updated schedule is output, called the forward-moved schedule, which is the basis of Step 3.

Step 3: backward move (BM)

Using the forward-moved schedule as the basis, BM performs same 4 procedures as Step 2 in the opposite way to delay activities as late as possible with the backward activity list. The list is constructed based on the descending order of activity finish times in the forward-moved schedule. Besides, in Step 3, only activities with negative CFs are considered in procedure 1. And in procedure 2, successors instead of predecessors are considered to determine the maximum allowable backward shift α_B, and in this step $t_{ext} = deadline$.

Once the traversal with BM is completed, the new forward-backward scheduling technique is finished, and the final optimized schedule is obtained.

3 Preliminary experiments and conclusions

To validate the performance of the forward-backward scheduling technique, existing research [Leyman and Vanhoucke, 2015] is adopted as the benchmark, and preliminary experiments using rescheduling strategy 1 are conducted on

small-sized instances from DC2 dataset [Vanhoucke, 2010]. Initial schedules are generated with 100 randomly created priority lists. The deadline is set based on the best known heuristic solutions [Debels and Vanhoucke, 2007]. The activity CFs vary between -50 and 50, and the discount rate is 1%. Results under 2 different deadlines and 3 different CF distributions are summarized in Table 1.

Table 1. Computational Results

Scenario	**%Improvement**	**%Better**	**%Equal**	**%Worse**
Deadline				
best known heuristic solutions + 5	19.28%	69.30%	8.26%	22.24%
best known heuristic solutions + 10	20.50%	66.53%	9.05%	24.42%
Cash flow distribution				
30% negative cash flows	7.05%	78.41%	0.18%	21.40%
60% negative cash flows	48.87%	69.49%	6.36%	24.12%
90% negative cash flows	2.19%	56.87%	18.37%	24.76%
Overall	19.95%	67.78%	8.70%	23.52%

%Improvement: percentage improvement over existing research.
%Better, %Equal, %Worse: percentage of optimized solutions that are better, equal, or worse relative to existing research.

As shown in Table 1, even with the base rescheduling strategy, the new technique outperforms existing methods by 19.95% on average, with 67.78% of its optimized solutions being superior. Its effectiveness is pronounced in the more realistic scenario where 60% of activities have negative CFs, yielding an improvement of 48.87%. Even under extreme scenarios, the technique maintains its advantage, achieving improvements of 7.05% and 2.19%, respectively. Therefore, it can be concluded that the newly proposed scheduling technique represents an alternative to existing research.

References

[Artigues et al., 2025] Artigues, C., Hartmann, S., and Vanhoucke, M. (2025). Fifty years of research on resource-constrained project scheduling explored from different perspectives. *European Journal of Operational Research*.

[Debels and Vanhoucke, 2007] Debels, D. and Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457–469.

[Leyman and Vanhoucke, 2015] Leyman, P. and Vanhoucke, M. (2015). A new scheduling technique for the resource–constrained project scheduling problem with discounted cash flows. *International Journal of Production Research*, 53(9):2771–2786.

[Vanhoucke, 2010] Vanhoucke, M. (2010). A scatter search heuristic for maximising the net present value of a resource-constrained project with fixed activity cash flows. *International Journal of Production Research*, 48(7):1983–2001.

Heuristic approaches for solving a bilevel parallel machine scheduling problem

Schau Q.[1,2],Della Croce F.[2],Ploton O.[1],T'kindt V.[1]

[1] Université de Tours, Laboratoire d'Informatique Fondamentale et Appliquée, Tours, France,
{quentin.schau,olivier.ploton,tkindt}@univ-tours.fr
[2] DIGEP, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy,
{quentin.schau,federico.dellacroce}@polito.it

Mots-Clefs. Scheduling, bilevel optimization, parallel machines, heuristics

1 Introduction

Bilevel scheduling is a scientific field at the intersection of the theory of scheduling and bilevel optimization. Bilevel scheduling problems involve processing a set of jobs on a set of resources, leading to a set of decisions. This set of decisions is partitioned between two agents: the leader's set of decisions X^L and the follower's set of decisions X^F. The leader makes the first move by making a decision $x^L \in X^L$, attempting to minimize its own objective function f^L. Next, the follower observes the leader's decision and responds by choosing a decision $x^F \in X^F$ to minimize its own objective function f^F. This results in a hierarchical and iterative decision-making process during which both objective functions can only be evaluated after all decisions are taken.
For some bilevel scheduling problems, the set of solutions to the follower's problem may contain more than one solution. In this case, the follower's decisions do not affect the value of its objective function f^F, but it does impact the value of the leader's objective function f^L. Thus, to provide a well-defined optimization framework for such problems, we need to define several scenarios. The optimistic case (OPT) corresponds to the scenario where the follower chooses a solution from its set of optimal solutions that leads to the best value of the leader's objective function. On the contrary, the pessimistic case (PES) corresponds to the scenario where the follower chooses a solution from its set of optimal solutions that is the worst for the leader's objective function.
To the best of our knowledge, the literature on bilevel scheduling problems is relatively limited. Specifically, exact algorithms are considered in (Abass 2005, Brown et al. 2005, Karlof and Wang 1996, Kovács and Kis 2011, T'kindt et al. 2024) and heuristic approaches are considered in (Bianco et al. 2015, Kis and Kovács 2012, Konur and Golias 2013, Lukač et al. 2008) for various bilevel scheduling problems.
To solve a bilevel scheduling problem heuristically, we explore the leader's set of decisions. Given a leader's decision $x^L \in X^L$, then the follower is expected to solve correspondingly its own problem. Note that this can be challenging when the follower's problem is $\mathcal{NP}$-hard.
This study focuses on a scenario with one leader and one follower composed of uniform parallel machines operating at two speeds. The leader selects the set of jobs to be scheduled, and aims to minimize the weighted number of tardy jobs. Then, the follower schedules the set of jobs provided by the leader, and aims to minimize the sum of completion times. From a bilevel perspective, we consider the optimistic case.
Unlike usual single-level methods, the bilevel approach enables the modeling of hierarchical decision-making processes within manufacturing systems. In such a system, a leader agent makes initial decisions, such as dispatching production orders, to

subordinate follower agents who are then responsible for scheduling those orders. Moreover, we can integrate some maintenance decisions into the scheduling process by requiring that potentially defective machines operate at their minimum speed, while healthy machines work at their maximum speed.

2 Definition and properties

The bilevel scheduling problem can be stated as follows: a set $\mathcal{J}$ of N jobs is given, where $n < N$ jobs must be selected (by the leader) to be scheduled (by the follower) on m_1 machines with high speed V_1 and on m_0 machines with low speed V_0. Each job J_j is characterized by its processing time p_j, weight w_j, and due date d_j. The follower's objective is to minimize the sum of completion times $\sum_j C_j^F$ where C_j^F denotes the completion time of job J_j. The leader's objective is to minimize the weighted number of tardy jobs $\sum_j w_j U_j^L$ where $U_j^L = 1$ if the job J_j is late, and $U_j^L = 0$ otherwise. Let $\mathfrak{S}_\mathcal{I}$ be the set of machine schedules that contain n jobs from a set of jobs $\mathcal{I}$ from $\mathcal{J}$, and let $C_j^F(s)$ be the completion time of job J_j in schedule s. Moreover, the optimistic scenario implies that there is a lexicographical objective function $Lex\left(\sum_j C_j^F, \sum_j w_j U_j^L\right)$, where we seek a schedule that minimizes the leader's objective function among all optimal schedules for the follower's problem. The bilevel problem can be formulated as:

$$\begin{array}{ll} \min\limits_{\substack{\mathcal{I}\subset\mathcal{J}\\|\mathcal{I}|=n}} & \sum\limits_{j\in\mathcal{I}} w_j U_j^L(s^*) \\ \text{st.} & s^* \in \operatorname*{argmin}\limits_{s\in\mathfrak{S}_\mathcal{I}} \left(Lex\left(\sum\limits_{j\in s} C_j^F(s), \sum\limits_{j\in s} w_j U_j^L(s) \right) \right) \end{array} \tag{1}$$

The follower's problem is a special case of the $Q||\sum_j C_j$ problem, which can be solved in $\mathcal{O}(n \log n)$ time using the Shortest Processing Time on First Available Machine (SPT-FAM) rule (Conway et al. 1967). Let $\mathcal{B}_b$ denote a block, defined as a set of pairs specifying a machine index and a position on machine schedule. A feasible schedule in this bilevel setting corresponds to a sequence of such blocks, where the processing times of all jobs associated with block $\mathcal{B}_b$ are strictly smaller than those associated with $\mathcal{B}_{b+1}$. This bilevel problem is denoted by $Q|OPT - n, V_i \in V_0, V_{max}|\sum_j C_j^F, \sum_j w_j U_j^L$, and is known to be $\mathcal{NP}$-hard in the strong sense (Schau et al. 2024). Exact algorithms were developed in (Schau et al. 2024, Schau et al. 2025). These methods can solve to optimality only instances with up to 40 jobs for 2 or 4 machines.

3 Heuristic approaches

To solve an optimistic bilevel minimization problem heuristically, two theoretical classes of heuristics can be considered.
The first class (1) operates only on the leader's decisions: given a leader decision, it solves the lexicographical problem optimally. This approach is the most common in the literature. However, for our problem, it leads to solving an $\mathcal{NP}$-hard problem. The second class (2) relaxes the requirements of class (1) by imposing optimality only on the first criterion of the lexicographical problem, namely $\sum_j C_j^F$, which is solvable in polynomial time. Given a leader's decision, it minimizes only the follower's objective function using the block structure property that guarantees optimality for $\sum_j C_j^F$, and then attempts to determine the best objective value for the leader's objective $\sum_j w_j U_j^L$. Consequently, this approach yields only an upper bound on the leader's objective function. To our knowledge, this contribution represents the first heuristic of this class within the bilevel scheduling literature.
We propose a Multi-Start Local Search (MSLS) heuristic, which uses a list of K initial solutions. This list is populated by a Recovering Beam Search (RBS) algorithm, and a Local Search (LS) is applied to each solution in the list at the end of

the RBS. Our heuristics exploit the block structure property to ensure optimality for the follower's problem. In this context, we determine two key elements: (1) the subset of jobs $\mathcal{I}$ that the leader selects, and (2) from this subset, the schedule w.r.t the block structure.
The LS begins with an initial solution and seeks to improve it by iteratively refining the leader's decisions and, afterwards, the follower's decisions (i.e., the schedule). The neighborhood of the leader's decisions is constructed by swapping a selected job with a non-selected job. This swap generates a new schedule, from which the neighborhood of the follower's decisions is built by identifying an improving swap of jobs within a block and then proceeding to the next block. The LS converges to a local optimum within a fast computational time. The primary challenge lies in devising a method for exploring diverse solutions in the solution space to reach a promising local optimum.
The RBS facilitates this exploration by employing the branching scheme developed in (Schau et al. 2025). Each node $\mathcal{N}$ of the search tree comprises a partial schedule and a set of removed jobs. The branching scheme proceeds by considering the next undecided job J_j according to the Shortest Processing Time order. The leader's decisions are then considered: either the job J_j is not selected (and thus removed in the corresponding child node $\mathcal{N}_c$); or the follower's decisions are considered, i.e., J_j is selected and assigned to one of the available locations in the first available block from the parent node, creating the corresponding child node $\mathcal{N}_c$. The RBS retains a subset of w child nodes with the highest evaluation values, using a function $f(\mathcal{N})$ that is a linear combination of the lower bound, as developed in (Schau et al. 2025), and an upper bound obtained by completing the partial schedule through successive assignment problems on each block. The upper bound gives a solution $S_{\mathcal{N}}$ for node $\mathcal{N}$. By retaining only w child nodes, the RBS achieves a sub-optimal exploration of the solution space. To recover from potential bad selection of nodes, a recovering mechanism is introduced. For a given node $\mathcal{N}$, decisions regarding scheduled and removed jobs are reassessed to reach a new node $\mathcal{N}'$ at the same level in the search tree. A heuristic condition is implemented in the recovery process: from a node $\mathcal{N}$, a new feasible solution $S'_{\mathcal{N}}$ is generated heuristically. If the value of $S'_{\mathcal{N}}$ is less than that of $S_{\mathcal{N}}$, a new node $\mathcal{N}'$ is created based on the decisions leading to $S'_{\mathcal{N}}$.
Finally, the list of initial solutions is populated with solutions $S_{\mathcal{N}}$ computed by the upper bound and $S'_{\mathcal{N}}$ obtained during the recovering phase, retaining only the $\max(0, K - w)$ best solutions. Additionally, the $\min(K, w)$ best solutions obtained from the leaf nodes of the search tree are included. Through the RBS branching scheme and the recovering phase, the solution space is effectively partitioned, enabling an efficient sampling of the space and ensuring that each solution is sufficiently distinct from the others. Once the RBS process is completed or if a time limit is reached, the LS is executed for all initial solutions, facilitating convergence to the closest local optimal solution.

4 Results and Conclusions

Instances were randomly generated, in which there were m_1 high-speed machines with their speed set at $V_1 = 2$, and m_0 low-speed machines with their speed set at $V_0 = 1$. The processing times were drawn at random from the uniform distribution $U[1, 100]$ and weights from the uniform distribution $U[1, 10]$. Due dates were generated using the uniform distribution $U[P \times (1 - tf - rdd/2), P \times (1 - tf + rdd/2)]$ with $tf, rdd \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ and $P = \frac{\sum_{j=1}^{N} p_j}{m_1 \times V_1 + m_0 \times V_0}$. Here, tf and rdd define a class of instances and we have 25 classes of instances. For each class, 10 instances of size $N = 80$ were randomly generated, leading to 1500 instances. For each value of N we set $n = N/4, N/2, 3N/4$, where values were rounded down in case they were fractional. Moreover, we set the total number of machines to $m = 2$, where we have the same number of high-speed and low-speed machines. These instances are split

into two categories: (A), composed of 860 instances where the Branch-and-Bound (BaB) developed in (Schau et al. 2025) finds an optimal solution, and (B), composed of 640 instances where BaB does not solve them to optimality. All methods were run with a time budget of 60 seconds.
Let $O_{\mathcal{M}}$ denote the sum of objective function values from method $\mathcal{M}$. The deviation is defined as: $\Delta = (O_{\mathcal{M}}^{\text{best}} - O_{\mathcal{M}})/O_{\mathcal{M}}^{\text{best}}$, it is negative when the $O_{\mathcal{M}}^{\text{best}}$ method is better, with a GAP of Δ% compared to the $O_{\mathcal{M}}$ method. Preliminary results show that the best configurations are: do not use the lower bound in the RBS and $k = 100$ for the MSLS.
Tables 1 and 2 show that the LS is very fast, while the MSLS is the slowest method. Our best approach is the MSLS with $w = 5$ on instances (B), with a GAP of -48.6% compared to BaB. For (A) instances, BaB gives better solutions with a GAP of -4.20% compared to MSLS. We remark that, in case much larger instances need to be considered, MSLS with w = 5 would not be the best choice: indeed, MSLS with w = 1 would allow us to guarantee still high quality results within a much limited CPU time effort.
To conclude, our novel approach to the optimistic bilevel scheduling problem, wherein we guarantee only the optimality of the first criterion in the lexicographical problem, has led to the development of effective heuristics.

Method	T_{avg}	$T_{\max}$	$O_{\mathcal{M}}$	Δ (%)
BaB60	58.52	60.91	22619	-48.6
LS_a	0.00	0.00	17886	-17.51
LS	0.27	0.53	16584	-8.95
RBS $w = 1$	3.62	6.21	15748	-3.46
RBS $w = 5$	17.69	31.09	15400	-1.18
MSLS $w = 1$	7.19	20.62	15395	-1.14
MSLS $w = 5$	21.33	34.27	15221	0.0

Table 1. Statistics of heuristics on (B) instances

Method	T_{avg}	$T_{\max}$	$O_{\mathcal{M}}$	Δ (%)
LS_a	0.00	0.00	1768	-85.71
LS	0.09	0.52	1539	-61.66
RBS $w = 1$	0.53	4.24	1006	-5.67
MSLS $w = 1$	2.08	7.84	1000	-5.04
MSLS $w = 5$	3.85	20.97	992	-4.20
RBS $w = 5$	2.21	18.84	976	-2.52
BaB60	4.71	58.78	952	0.0

Table 2. Statistics of heuristics on (A) instances

References

Abass, S.A. Bilevel programming approach applied to the flow shop scheduling problem under fuzziness, *Computational Management Science*, Vol. 2, pp. 279-293.

Bianco, L., Caramia, M., Giordani, S., Mari, R. Grid scheduling by bilevel programming A heuristic approach, *European Journal of Industrial Engineering*, Vol. 9, pp. 101-125.

Brown, G.G., Carlyle, W.M., Royset, J.O., Wood, R.K. On the complexity of delaying an adversary's project, *The Next Wave in Computing, Optimization, and Decision Technologies*

Colson, B., Marcotte, P., Savard, G. Bilevel programming A survey, *4OR*, Vol. 3, pp. 87-107.

Conway, R.W., Maxwell, W.L., Miller, L.W. Theory of Scheduling. Addison-Wesley, 1967.

Karlof, J.K., Wang, W. Bilevel programming applied to the flow shop scheduling problem, *Computers & Operations Research*, Vol. 23, pp. 443-451.

Kis, T., Kovács, A. On bilevel machine scheduling problems, *OR Spectrum*, Vol. 34, pp. 43-68.

Kovács, A., Kis, T. Constraint programming approach to a bilevel scheduling problem, *Constraints*, Vol. 16, pp. 317-340.

Konur, D., Golias, M.M. Analysis of approaches to cross-dock truck scheduling with truck arrival time uncertainty, *Computers & Industrial Engineering*, Vol. 65, pp. 663-672.

Lukač, Z., Šorić, K., Rosenzweig, V.V. Production planning problem with sequence-dependent setups as a bilevel programming problem, *European Journal of Operational Research*, Vol. 187, pp. 1504-1512.

Schau, Q., Ploton, O., T'kindt, V., Della Croce, F. Bilevel scheduling of uniform parallel machines in the context of coupling maintenance and scheduling decisions, *PMS 2024*.

Schau, Q., Ploton, O., T'kindt, V., Della Croce, F., Hoogeveen, H. Exact algorithms for the bilevel scheduling of uniform parallel machines with coupled maintenance and production decisions, *MAPSP 2024*.

Schau, Q., Ploton, O., T'kindt, V., Della Croce, F., Hoogeveen, H., Hoogeveen, J. Exact algorithm for coupling maintenance-production: a bilevel approach, *CIGI QUALITA MOSIM 2025*.

T'kindt, V., Della Croce, F., Agnetis, A. Single machine adversarial bilevel scheduling problems, *European Journal of Operational Research*, Vol. 315, pp. 63-72.

Initial-Solution Sampling for the Job Shop Scheduling Problem : Fitness-Landscape Analysis

Essognim Richard WILOUWOU[1], Arwa KHANNOUSSI[2], Alexandru-Liviu OLTEANU[1] and Marc SEVAUX[1]

[1] University of South Brittany, France
essognim.wilouwou, alexandru.olteanu, marc.sevaux@univ-ubs.fr
[2] IMT Atlantique, France
arwa.khannoussi@imt-atlantique.fr

Keywords: Job Shop Scheduling Problem, Metaheuristics, Variable Neighborhood Search, Fitness Landscape Analysis, Machine Learning.

1 Introduction

Scheduling is an optimization problem frequently encountered in manufacturing systems. The most commonly observed in practice is Job Shop Scheduling Problem (JSSP). This problem has been proven to be $\mathcal{NP}$-hard, which makes exact methods impractical for real-world applications. Indeed, in order to remain competitive, companies require methods that are not computationally expensive while still providing solutions of acceptable quality. This naturally leads to the use of approximate methods. These approaches fall into two main categories: heuristics and metaheuristics. In this work, we focus on metaheuristics approaches, which generally rely on three main components: initialization, intensification, and diversification. It has been shown that most metaheuristics are highly sensitive to their initialization (Li, Liu and Yang 2020).

In this work, we will focus on the initialization phase improvement. Machine learning models have demonstrated strong capabilities in capturing and modeling complex structures and have been widely adopted in the literature to enhance the performance of metaheuristics (Talbi 2021),(Zhang, Wang, Liu, Su, Ishibuchi and Jin 2025). Given a set of candidate initial solutions, we use a trained binary classifier to predict whether a solution is located in the *most promising regions* of the search landscape relative to the others, based on landscape features extracted around each solution. The predicted promising solution is then selected to initialize our optimization algorithm. As optimization approach we use the simple version of the Variable Neighborhood Search (SVNS) which principles and applications are discussed in (Hansen, Mladenović, Brimberg and Pérez 2018). In addition, the learned model can be used to categorize problem instances according to their difficulty.

2 Fitness landscape analysis

Several features have been proposed in the literature to capture useful information about the landscape surrounding a solution (Angel and Zissimopoulos 1998, Pitzer and Affenzeller 2012). These include ruggedness, modality, fitness-distance correlation, autocorrelation, correlation length, size of the basin of attraction, distribution of objective function values. These features are extracted by performing random walks in the solution space.

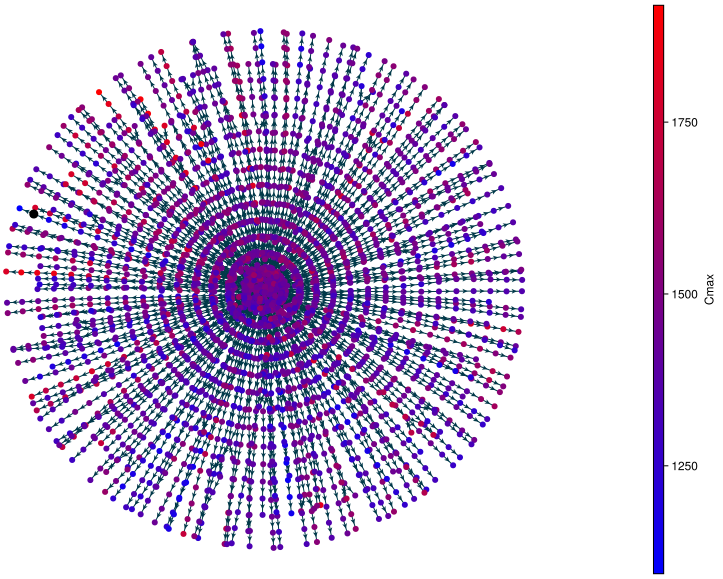


Fig. 1. Random walks, Instance *LA18*

In Fig. 1, we illustrate real random walks around a solution. This type of visualization was introduced in (Ochoa, Malan and Blum 2021) to study the behavior of metaheuristics for the Traveling Salesman Problem. In this representation, each node corresponds to a solution, the central node represents the starting solution, and the color of each node is proportional to its objective function value.

3 Resolution Approach

Our methodology combines an analysis of the solution landscape with supervised learning techniques to guide the selection of the most promising solution. We first extract local and relative features for each solution, then use these features to predict whether a given solution is relatively more promising than others. As illustrated in Figure 2, given an instance of the problem, our approach first samples a set of initial solutions, $\mathcal{S}$. A filtering procedure based on our binary classifier is then applied to retain only the solutions considered relatively promising. The most promising solution after filtering is used to initialize our SVNS. If multiple solutions are deemed promising, one is randomly selected to initialize the SVNS.

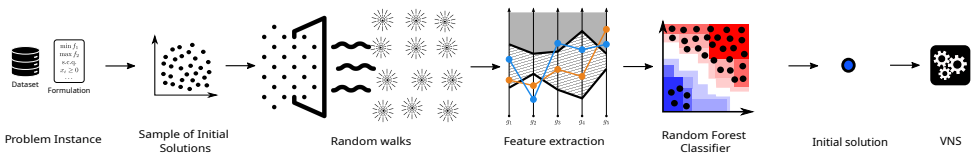


Fig. 2. Description of our approach

4 Numerical tests

Experimental design: To evaluate the performance of the proposed approaches, numerical experiments were conducted on well-known benchmarks, including *FT06-10-20* (Muth, Thompson and Winters 1963), *LA01-40* (Lawrence 1984), *ABZ5-9* (Adams, Balas and Zawack 1988), and *ORB01-10* (Applegate and Cook 1990). To evaluate the performance of our method, we use the gap to the optimum (1) as the performance metric.

$$Gap = 100 \times \frac{C_{\max} - C^*_{\max}}{C^*_{\max}} \tag{1}$$

Two experimental results are presented in this section: learning performance and solving performance. To evaluate the solving performance of our approach *(VNS/RF)*, we compare it against four classical initialization strategies: *Random Sampling (VNS/Random)*, *Most Work Remaining (VNS/MWR)*, *Most Operations Remaining (VNS/MOR)*, *First In First Out (VNS/FIFO)*. Each of these strategies achieves better performance than the others on a subset of instances. The objective of our approach is to use our trained model to select the most effective initialization strategy for each instance among these four classic strategies.

Learning Performance: In this work, we use *random forests* (RF) as the learning model for our task, setting the number of trees to 100 while keeping all other parameters at their default values. To evaluate learning performance, we use the F_1*-score* metric which is widely adopted for binary classification, as it combines two key measures: precision and recall.

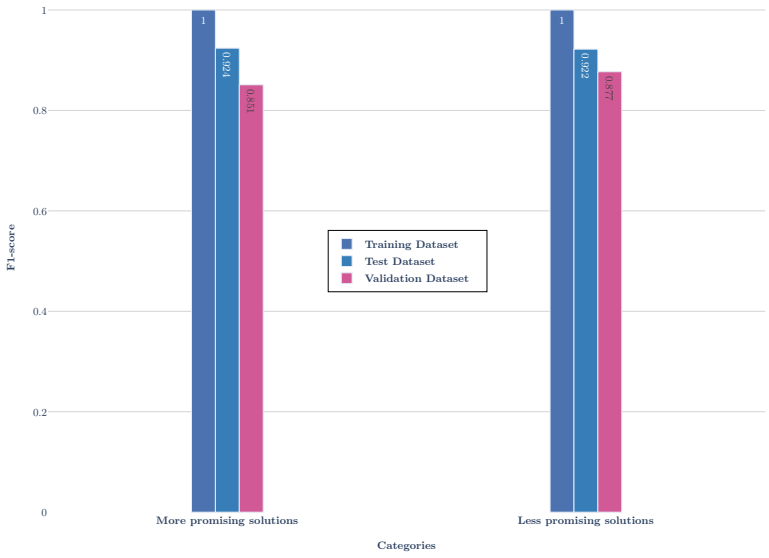


Fig. 3. F_1-score by considering each category as the positive class

The learning performance of our model demonstrates that our features are strongly correlated with the target prediction and that the random forest model is able to learn effectively, the F_1-*score* is above 85%.

Performance Evaluation: As shown in Table 1, on average, our approach achieves a lower mean gap for the majority of instances within a 5-minute time budget, including both initialization and optimization. The detailed results show that our model is able to make accurate predictions for the majority of instances. Although part of the time is spent on prediction, this overhead is still compensated by the overall performance gains.

	VNS/Random		**VNS/MWR**		**VNS/MOR**		**VNS/FIFO**		**VNS/RF**	
Benchmarks	AvgGap (%)	std	AvgGap (%)	std	AvgGap (%)	std	AvgGap (%)	std	AvgGap (%)	std
ft	6.97	6.04	5.52	5.48	9.11	8.25	4.04	3.84	**3.36**	3.1
la01-20	1.76	2.98	1.26	2.07	0.98	1.66	0.86	1.61	**0.72**	1.28
la21-30	2.8	3.78	2.01	2.66	1.61	2.06	1.54	2.08	**1.26**	1.58
la31-40	10.5	4.69	5.88	4.75	6.16	4.91	6.86	4.75	**5.46**	4.32
abz	16.37	13.35	12.92	10.87	12.62	9.55	**12.37**	9.14	12.66	10.1
orb	11.04	3.65	7.62	3.89	11.25	4.86	7.74	3.01	**7.09**	3.32
Average	8.24	-	5.87	-	6.96	-	5.57	-	**5.09**	-

Table 1. Average gap to the optimum

Acknowledgements

This work was conducted with the support of the Brittany Region in France.

References

Adams, J., Balas, E. and Zawack, D. (1988), 'The shifting bottleneck procedure for job shop scheduling', *Management science* **34**(3), 391–401.

Angel, E. and Zissimopoulos, V. (1998), 'Autocorrelation coefficient for the graph bipartitioning problem', *Theoretical Computer Science* **191**(1-2), 229–243.

Applegate, D. and Cook, W. (1990), *A computational study of job-shop scheduling*, School of Computer Science, Carnegie Mellon University.

Hansen, P., Mladenović, N., Brimberg, J. and Pérez, J. A. M. (2018), Variable neighborhood search, *in* 'Handbook of metaheuristics', Springer, pp. 57–97.

Lawrence, S. (1984), 'Resource constrained project scheduling', *an experimental investigation of heuristic scheduling techniques* .

Li, Q., Liu, S.-Y. and Yang, X.-S. (2020), 'Influence of initialization on the performance of metaheuristic optimizers', *Applied Soft Computing* **91**, 106193.

Muth, J. F., Thompson, G. L. and Winters, P. R. (1963), 'Industrial scheduling', *(No Title)* .

Ochoa, G., Malan, K. M. and Blum, C. (2021), 'Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics', *Applied Soft Computing* **109**, 107492.

Pitzer, E. and Affenzeller, M. (2012), 'A comprehensive survey on fitness landscape analysis', *Recent advances in intelligent engineering systems* pp. 161–191.

Talbi, E.-G. (2021), 'Machine learning into metaheuristics: A survey and taxonomy', *ACM Computing Surveys (CSUR)* **54**(6), 1–32.

Zhang, R., Wang, J., Liu, C., Su, K., Ishibuchi, H. and Jin, Y. (2025), 'Synergistic integration of metaheuristics and machine learning: latest advances and emerging trends', *Artificial Intelligence Review* **58**(9), 268.

Using Simulated Annealing for Solving Reentrant Permutation Flow Shop Problems

Itamar Segal[1], Tal Grinshpoun[1], Hagai Ilani[2] and Elad Shufan[2]

[1] Ariel University, Ariel, Israel
itamar.segal@msmail.ariel.ac.il, talgr@ariel.ac.il
[2] Shamoon College of Engineering, Ashdod, Israel
hagai@sce.ac.il, elads@sce.ac.il

Keywords: Permutation Flow Shop, Reentrant Flow Shop, Makespan, Simulated Annealing.

1 Introduction

A flow shop is a scheduling environment in which all jobs visit a set of machines in the same order. A well-studied special case is the *permutation* flow shop (PFS), where the sequence of jobs is identical across all machines. In a standard flow shop, each job visits each machine at most once. However, in many real-world manufacturing systems, jobs must return to the same machine multiple times during processing. Such problems are referred to as *reentrant* flow shops (RFS) [Danping and Lee, 2011].

A particular class of RFS is the *cyclic*-RFS, in which each job passes through all machines multiple times. A cyclic-RFS consists of n jobs, m machines, and l levels. Each level represents a full pass of a job through the machines in the order $(1, 2, \ldots, m)$. The processing of a job at a specific level is called a sub-job. RFS models arise in various manufacturing environments [Danping and Lee, 2011].

The notion of permutation has been extended to RFS, leading to the reentrant permutation flow shop (RPFS), primarily in the cyclic setting. In an earlier work [Segal et al., 2024], we showed that the concept of permutation was used inconsistently in the literature. To clarify its meaning, we defined four distinct RPFS types, based on the following constraints on sub-job scheduling order:

- Level passing: After completing a sub-job at level l^*, must all other sub-jobs of the same level be completed before any sub-job from a higher level ($l > l^*$) may start? If so, we say that level passing is prohibited, otherwise allowed.
- Job passing: Must the job order within a given level be identical in all other levels? If so, we say that job passing is prohibited, otherwise allowed.

Based on these constraints, four RPFS types are defined: PP (Passing Prohibited for both level and job passing), LPP (Level Passing Prohibited), JPP (Job Passing Prohibited), and PA (Passing Allowed for both level and job passing).

In a previous work, we developed several heuristics aimed at producing low-makespan schedules for each type [Segal et al., 2026]. The heuristics extend classical flow shop methods, including Palmer's slope rule [Palmer, 1965] and

NEH [Nawaz et al., 1983]. NEH-based variations generally performed well for the PP and LPP types. All heuristics yielded poor results for the PA and JPP types. In addition, several studies have investigated the PP [Chen et al., 2009] and PA [Xu et al., 2014] types of RPFS using metaheuristic frameworks. The present study focuses on obtaining high-quality solutions across the *four* permutation types by employing a simulated annealing (SA) metaheuristic.

2 Simulated Annealing Adaptation for RPFS Types

SA is known for its effectiveness in exploring large search spaces and producing high-quality solutions for complex optimization problems. In particular, SA has been applied extensively in flow-shop scheduling; see, for example, [Chen and Sandnes, 2015].

In this study, we adopted three standard perturbation techniques to generate candidate neighbors in PFS [Hurkała and Hurkała, 2012] and extended these to the four types of RPFS.

1. Adjacent swap: interchanging two adjacent jobs.
2. Global swap: interchanging two jobs at arbitrary positions.
3. Insert: removing a job and reinserting it elsewhere in the sequence.

Following are the extensions for the four RPFS types:

- **PP:** The three moves are implemented as in the standard PFS.
- **LPP:** The three moves are applied to sub-jobs within the same level, either (a) level by level (from level 1 to l) or (b) on a randomly selected level.
- **JPP:** The three moves are applied to the sub-job order, provided that precedence constraints are preserved; that is, level $k+1$ of a job does not precede level k of the same job, and a consistent job order is maintained across all levels. In addition, an adjacent-swap move is applied to the job order, resulting in four moves in total.
- **PA:** The three moves are applied to the sub-job order, provided that precedence constraints are preserved.

We used an initial temperature $T_0 = 100$, a final temperature $T_{\text{final}} = 0.1$, and a cooling rate $\alpha = 0.98$. At each temperature, $10 \cdot n$ neighbors were evaluated for the PP type and the level-by-level LPP, and $5 \cdot n \cdot l$ for the random-level LPP, JPP, and PA. The initial sequence was either a random permutation or a solution obtained by an NEH-based heuristic from our previous work [Segal et al., 2026].

The moves were selected uniformly at random for the PP, LPP, and PA types. For the JPP type, moves were selected according to a (0.3, 0.3, 0.3, 0.1) distribution. The job-order move was assigned a lower probability due to its major impact on the sequence.

3 Experimental Evaluation

Computational experiments were conducted to evaluate the SA mechanism described in Section 2 across all four permutation types. Experimental sets are

denoted by $n \times m \times l$, with processing times drawn from a discrete uniform distribution over $[1, 19]$. For each set, 50 instances were randomly generated and evaluated. As a first step, we compared the SA results with optimal solutions for a relatively small $6 \times 5 \times 2$ set, for which optimality can be obtained by exhaustive search. Let $C_{\max}(\mathrm{OPT}, X)$ denote the optimal makespan and $C_{\max}(\mathrm{SA}, X)$ denote the SA-calculated makespan for permutation type $X \in \{PP, LPP, JPP, PA\}$. The performance gap of an instance with respect to a permutation type X for this step is defined as

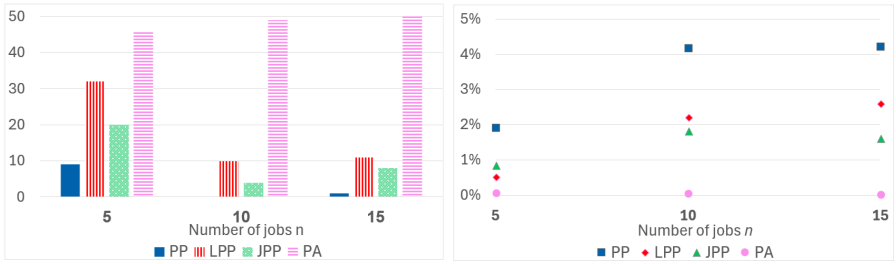
$$\text{Performance gap}(X) = \frac{C_{\max}(\mathrm{SA}, X) - C_{\max}(\mathrm{OPT}, X)}{C_{\max}(\mathrm{OPT}, X)} \times 100.$$

SA achieved the optimal makespan in 100%, 98%, 90%, and 92% of the instances for the PP, LPP, JPP, and PA types, respectively, with an average performance gap below 0.06% and a maximum gap of 1%.

In the second step, we ran the SA mechanism on larger sets, with a base problem size of $10 \times 5 \times 3$. The results are shown in Figures 1 and 2, corresponding to variations in n and l, respectively. Each figure consists of two graphs: the left-hand graphs report the permutation types achieving the best makespan in each set (with counts possibly exceeding 50 due to ties), while the right-hand graphs present the average performance gap for each permutation type. The performance gap for a permutation type X is defined here as

$$\text{Performance gap}(X) = \frac{C_{\max}(SA, X) - C_{\max}(\mathrm{Best})}{C_{\max}(\mathrm{Best})} \times 100,$$

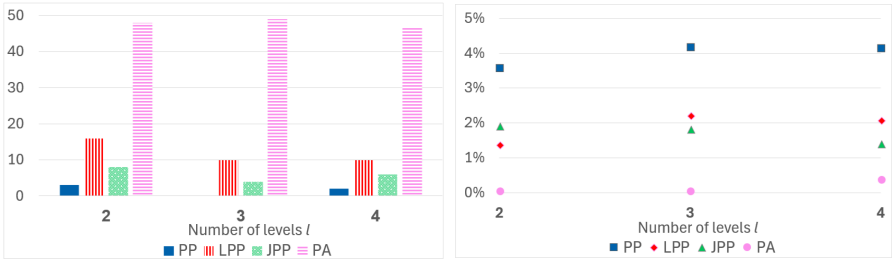
where $C_{\max}(\mathrm{Best}) = \min_X C_{\max}(SA, X)$.



(a) Number of best makespans (b) Average performance gap (%)

Fig. 1: Varying n, with $m = 5$ and $l = 3$.

The results are as expected for a well-performing algorithm and indicate that, in most instances, SA consistently finds the best solution within the PA solution space, which contains all other permutation solution spaces. The PP type yields the smallest number of best solutions, while the numbers obtained for the LPP and JPP types lie between those of PA and PP. The average performance gap



(a) Number of best makespans (b) Average performance gap (%)

Fig. 2: Varying l, with $m = 5$ and $n = 10$.

shows the same trend. These results contrast with the heuristics outcomes, as the heuristics exhibited difficulties in finding the best solutions, especially within the PA and JPP solution spaces. As part of our ongoing investigation, we are evaluating the SA mechanism on additional experimental sets and exploring alternative SA parameter configurations.

References

[Chen et al., 2009] Chen, J.-S., Pan, J. C.-H., Lin, C.-M., et al. (2009). Solving the reentrant permutation flow-shop scheduling problem with a hybrid genetic algorithm. *International Journal of Industrial Engineering*, 16(1):23–31.

[Chen and Sandnes, 2015] Chen, R.-M. and Sandnes, F. E. (2015). Flow shop scheduling based on a novel cooling temperature driven simulated annealing algorithm.

[Danping and Lee, 2011] Danping, L. and Lee, C. K. (2011). A review of the research methodology for the re-entrant scheduling problem. *International Journal of Production Research*, 49(8):2221–2242.

[Hurkała and Hurkała, 2012] Hurkała, J. and Hurkała, A. (2012). Effective design of the simulated annealing algorithm for the flowshop problem with minimum makespan criterion. *Journal of Telecommunications and Information Technology*, pages 92–98.

[Nawaz et al., 1983] Nawaz, M., Enscore Jr, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

[Palmer, 1965] Palmer, D. S. (1965). Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. *Journal of the Operational Research Society*, 16(1):101–107.

[Segal et al., 2024] Segal, I., Grinshpoun, T., Ilani, H., and Shufan, E. (2024). The meaning of permutation in reentrant permutation flow shop problems. In *PATAT 2024: 14th International Conference on the Practice and Theory of Automated Timetabling*, pages 99–109.

[Segal et al., 2026] Segal, I., Grinshpoun, T., Ilani, H., and Shufan, E. (2026). Reentrant permutation flow shop problems: Definitions and heuristics. Accepted to Journal of Scheduling.

[Xu et al., 2014] Xu, J., Yin, Y., Cheng, T. C. E., Wu, C.-C., and Gu, S. (2014). A memetic algorithm for the re-entrant permutation flowshop scheduling problem to minimize the makespan. *Applied Soft Computing*, 24:277–283.

E - Machine Scheduling #1

Fairness in Repetitive Scheduling - Survey of Results and Future Challenges

Danny Hermelin[1] and Dvir Shabtay[1]

Ben Gurion University of the Negev, Israel
hermelin, dvirs@bgu.ac.il

Keywords: Repetitive scheduling, Algorithmic fairness, Quality of service.

1 Introduction

In many scheduling problems, jobs are associated with clients, and any solution may have a different effect on the quality of service (QoS) received by each client. Traditional objectives have primarily focused on maximizing system efficiency. However, such objectives may lead to solutions that create substantial disparities in QoS across clients, which can be perceived as unfair and may ultimately result in the loss of future customers.

Our research focuses on a class of repetitive scheduling problems in which the same set of clients repeatedly submit jobs to a scheduling system over a finite sequence of periods (e.g., days). In such settings, clients who receive a poor schedule on one day may be compensated with a better schedule on other days, ensuring that their overall QoS across the scheduling horizon does not become excessively low. Accordingly, our main goal is to demonstrate how the repetitive nature of these problems can be leveraged to construct a sequence of schedules that, when considered together, improves fairness.

1.1 Problem definition

The class of repetitive scheduling problems we consider has been formally presented in (Hermelin *et. al.* 2025) and is defined below. We are given a set of n *clients* ($j \in [n] := \{1, \ldots, n\}$). Each client is associated with a single job on each of q different *days*. By (i, j) we denote the job of client j on day i. On each day $i \in [q]$, the jobs in $\mathcal{J}_i^d = \{(i, j) : j \in \{1, \ldots, n\}\}$ have to be scheduled on a single machine. While the model can be easily extended to more general machine environments, we focus here on the single-machine case.

An instance of our problem includes a set of job processing times $\{p_{i,j} : i \in [q], j \in [n]\}$, where $p_{i,j}$ denotes the *processing time* of job (i, j) on the single machine. When relevant, the instance may also include additional parameters for each job (i, j) such as its *due date* $d_{i,j}$ or *weight* $w_{i,j}$. A solution π to the problem consists of a set of q daily schedules, $\pi = (\pi_1, \ldots, \pi_q)$, where $\pi_i = (\pi_i(1), \ldots, \pi_i(n))$ denotes the job processing permutation on day i and $\pi_i(j)$ the position in which job (i, j) is processed on that day. Given a solution, we assume that the completion times $\{C_{i,j}(\pi) : i \in [q], j \in [n]\}$ can be easily computed. For example, when the objective is *regular* (i.e., a nondecreasing function of the job completion times), the completion time of job (i, j) can be computed by $C_{i,j}(\pi) = \sum_{\ell \in [n]: \pi_i(\ell) \leq \pi_i(j)} p_{i,\ell}$.

Given a solution, let $F_{i,j}(\pi)$ denote the *performance measure* of job (i, j) indicating the QoS that client j receives on day i. Common examples of performance measures include (i) the completion time $C_{i,j}(\pi)$ of job (i, j); (ii) the lateness $L_{i,j}(\pi) = C_{i,j}(\pi) - d_{i,j}$ of job (i, j); (iii) and the *binary tardiness indicator* $U_{i,j}(\pi)$ of job (i, j), where $U_{i,j}(\pi) = 1$ iff $C_{i,j}(\pi) > d_{i,j}$. With this notation, the QoS received by client j across all q days is $F_j(\pi) = \sum_{i \in [q]} F_{i,j}(\pi)$, where, in most cases, a smaller value indicates higher QoS.

1.2 Efficiency via Fairness

Following (Hermelin *et. al.* 2025), we consider two criteria for evaluating the quality of a solution π. The first criterion, denoted by $F(\pi)$, is the *fairness* measure:

$$F(\pi) = \max_{j\in[n]} F_j(\pi) = \max_{j\in[n]} \sum_{i\in[q]} F_{i,j}(\pi). \tag{1}$$

The second criterion, denoted by $G(\pi)$, is the *efficiency* measure:

$$G(\pi) = \sum_{j\in[n]} F_j(\pi) = \sum_{i\in[q]} \sum_{j\in[n]} F_{i,j}(\pi). \tag{2}$$

(Hermelin *et. al.* 2025) defined two variants of the problem, each optimizing one of the two criteria. The first, the *Maximal Fairness* problem, aims to find a schedule π that minimizes $F(\pi)$. The second, the *Maximal Efficiency* problem, seeks a schedule π that minimizes $G(\pi)$. Using the classical three-field notation for scheduling problems, we denote the Maximal Fairness problem by $1|rep|\max_j \sum_i F_{i,j}$, and the Maximal Efficiency problem by $1|rep|\sum_i \sum_j F_{i,j}$. As the Maximal Efficiency problem reduces to q independent (single day) $1||\sum_j F_{i,j}$ problems, one for each $i \in [q]$, the following corollary holds:

Corollary 1. *The Maximal Efficiency problem,* $1|rep|\sum_i \sum_j F_{i,j}$*, is solvable in* $O(qh(n))$ *time, where* $O(h(n))$ *is the time required to solve the single-day counterpart,* $1||\sum_j F_{i,j}$*.*

Corollary 1 above implies that if the classical (single day) problem is solvable in polynomial time, so is the Maximal Efficiency problem. For example, the fact that $1||\sum_j F_{i,j}$ is solvable in $O(n \log n)$ time when $F_{i,j} \in \{C_{i,j}, L_{i,j}, U_{i,j}\}$ (Smith 1956, Moore 1968) implies that the corresponding $1|rep|\sum_i \sum_j F_{i,j}$ is solvable in $O(qn \log n)$ time. This observation also enables to construct lower bound for the minimal value of the Maximal Fairness problem using the optimal solution for the Maximal Efficiency problem. To do so, let π^F and π^G be the optimal solutions for the Maximal Fairness and the Maximal Efficiency problems, respectively. It is easy to see that the following holds $F(\pi^F) \geq G(\pi^F)/n \geq G(\pi^G)/n$. The first inequality follows from the fact that the maximum over a set of numbers is at least the average value of this set of numbers, and the second inequality holds due to the optimality of π^G for the Maximal Efficiency problem. Therefore, we have the following corollary:

Corollary 2. $G(\pi^G)/n$ *provides a lower bound on the minimum solution value of the Maximal Fairness problem.*

Due to space limitations, we focus on presenting the key results for the Maximal Fairness problem with $F_{i,j} = C_{i,j}$. The main approximation results have not yet been published, while all other results are taken from (Hermelin *et. al.* 2025, Plotkin *et. al.* 2025). For results on other performance measures, we refer to (Heeger *et. al.* 2023, Heeger *et. al.* 2025, Hermelin *et. al.* 2025).

2 Hardness results and exact algorithms

We begin by presenting our main hardness result:

Theorem 1. *The* $1\,|rep|\max_j \sum_i C_{i,j}$ *problem is strongly NP-hard, and unless* P = NP*, it cannot be approximated in polynomial-time within factor smaller than* 38/37*.*

As the reduction leading to the above strong NP-hardness result involves an arbitrary number of clients and days, we now focus on cases where one of these parameters is small.

Consider first the case of few days. When $q = 2$, we proved that applying a procedure similar to the famous Johnson's algorithm (Johnson 1954) for minimizing the makespan in a two-machine flow-shop scheduling can optimally solve the Maximal Fairness problem. In fact, the algorithm constructs a job sequence similar to Johnson's algorithm, where the days act as machines. However, it reverses the schedule for the second day, whereas in Johnson's algorithm, the optimal schedule is a permutation schedule, meaning that the optimal job permutation is the same for both machines.

Theorem 2. *The* $1\,|rep|\max_j \sum_i C_{i,j}$ *problem is solvable in* $O(n \log n)$ *time when* $q = 2$.

The next question that trivially arises is: can we easily solve the problem also when there are more than two days? Unfortunately, the answer for this problem is no. In fact, by a polynomial time reduction from Partition we proved the following:

Theorem 3. *The* $1\,|rep|\max_j \sum_i C_{i,j}$ *problem is weakly NP-hard for all* $q \geq 4$.

Theorems 2 and 3 provide a nearly complete characterization of the hardness of the Maximal Fairness problem with respect to the parameter q, leaving the case of $q = 3$ unsolved. Consider now parameter n. We showed that the problem is (weakly) NP-hard already for $n = 2$ (but q being arbitrary). The reduction is again from Partition.

Theorem 4. *The* $1\,|rep|\max_j \sum_i C_{i,j}$ *problem is weakly NP-hard for* $n \geq 2$.

On the algorithmic side, the $1\,|rep|\max_j \sum_i C_{i,j}$ problem admits a fixed-parameter tractable (FPT) algorithm with respect to the combined parameter of $q + \bar{K}$, where $\bar{K}$ is an upper bound on the optimal solution value, i.e., an algorithm solving the problem in $f(q + \bar{K}) \cdot n^{O(1)}$ time for some computable function f. Moreover, the problem admits a $f(n + \bar{K})^{O(n)}$ time algorithm, which is pseudo-polynomial when the number of clients is constant ($n = O(1)$). Another interesting case is the unit processing times case, which can be solved easily using a simple algorithm. In fact, when q is even, any solution that reverses the permutations between even and odd days is optimal. When $q \geq 3$ is odd, the optimal solution coincides with the even-day structure for $q - 3$ of the days, while the remaining three days require more carefully designed permutations.

3 Approximability results

The inapproximability result in Theorem 1 rules out (assuming P $\neq$ NP) the existence of a polynomial-time approximation scheme (PTAS) for the problem. A first attempt to construct an approximation algorithm for the Maximal Fairness problem appears in (Plotkin *et. al.* 2025). They proposed a heuristic procedure that decomposes the problem into $\lfloor q/2 \rfloor$ two-day instances, each of which is optimally solved in $O(n \log n)$ time. The heuristic then combines these two-day solutions into a single solution for the entire problem (if q is odd, the last day is scheduled arbitrarily). This approach results in a $\lceil q/2 \rceil$-approximation, which is bounded only when $q = O(1)$. However, a more sophisticated (yet unpublished) algorithm, based on linear programming (LP) relaxation followed by a clever rounding procedure, enables us to obtain the following result:

Theorem 5. *The* $1|rep|\max_j \sum_i C_{i,j}$ *problem can be approximated within a factor of 2 in polynomial-time.*

When the number of n clients is constant, the $f(n + \bar{K})^{O(n)}$ time algorithm can be converted into a fully polynomial-time approximation scheme (FPTAS) by applying standard scaling ideas. However, when the number of days q is constant we were only able to design a polynomial-time approximation scheme (PTAS) for the problem. The PTAS relies on newly-devised batching scheme, where intuitively each batch represents a set of jobs that we conceptually group together and treat as being simultaneously completed.

Theorem 6. *For any $\epsilon > 0$, the $1|rep|\max_j \sum_i C_{i,j}$ problem can be approximated within factor of $1+\epsilon$ in fully polynomial time when n is constant, and in polynomial time when q is constant.*

4 The price of fairness

Solving the Maximal Fairness problem may lead a solution which is sub-optimal in terms of efficiency. The quantity that captures the relative loss of efficiency is known as the *Price of Fairness* (PoF) (Bertsimas *et. al.* 2011, Agnetis *et. al.* 2019). To formally define the PoF, consider a given instance I of our problem, and among all optimal solutions for the Maximal Fairness problem, let $\pi^F(I)$ be the least efficient one. Moreover, let $\pi^G(I)$ be an optimal solution for the Maximal Efficiency problem with instance I. Then, the PoF is simply the minimum value of δ which satisfies $\frac{G(\pi^F(I))}{G(\pi^G(I))} \leq \delta$ in all possible instances I of the problem.

Theorem 7. *The PoF of the $1\,|rep|\max_j \sum_{i=1}^{q} C_{i,j}$ problem is between $n/2$ and n.*

5 Open Problems

Although we were able to provide a variety of complexity and algorithmic results for the $1\,|rep|\max_j\{\sum_{i=1}^{q} C_{i,j}\}$ problem, many research questions remain open. We below present several examples:

- Can we solve the $q = 3$ case in polynomial time? What about the special case of daily-independent instances, where $p_{i,j} = p_i$?
- Can we solve the problem with constant number of days in pseudo polynomial time or is this case strongly NP-hard?
- Can we construct a constant-factor approximation algorithm with an approximation ratio less than 2?

There are many other open challenges to explore, particularly with respect to other performance measures and more complex machine environments.

References

Agnetis, A., B. Chen, G. Nicosia and A. Pacifici, 2019, "Price of fairness in two-agent single-machine scheduling problems", *European Journal of Operational Research*, Vol. 276, pp.79-87.

Bertsimas, D., V.F. Farias and N. Trichakis, 2011, "The price of fairness", *Operations Research*, Vol. 59, pp.17-31.

Heeger, K., D. Hermelin, G.B., Mertzios, H. Molter, R. Niedermeier and D. Shabtay, 2023, "Equitable scheduling on a single machine", *Journal of Scheduling*, Vol. 26, pp.209-225.

Heeger, K., D. Hermelin, Y. Itzhaki, H. Molter and D. Shabtay, 2023, "Fair repetitive interval scheduling", *Algorithmica*, Vol. 87, pp.1340-1368.

Hermelin, D., H. Molter, R. Niedermeier, M. Pinedo, and D. Shabtay, 2025, "Fairness in repetitive scheduling", *European Journal of Operational Research*, Vol. 323, pp. 724-738.

Johnson, S.M., 1954, "Optimal two and three-stage production schedules with setup times included", *Naval Research Logistics Quarterly*, Vol. 1, pp. 61-67.

Moore, J.M., 1968, "An n job, one machine sequencing algorithm for minimizing the number of late jobs", *Management Science*, Vol. 15, pp. 102-109.

Plotkin, A., Y. Fink and D. Shabtay, 2025, "Algorithms for Fair Repetitive Scheduling", *Computers and Industrial Engineering*, Vol. 212, pp. 111659.

Smith, W.E., 1956, "Various optimizers for single-stage production", *Naval Research Logistics Quarterly*, Vol. 3, pp. 59-66.

Optimistic bilevel scheduling with job selection on a single machine and maximum lateness minimization

T'kindt V.[1], Agnetis A.[2] and , Della Croce F.[3]

[1] Université de Tours, Laboratoire d'Informatique Fondamentale et Appliquée, Tours, France,
tkindt@univ-tours.fr
[2] Università degli Studi di Siena, Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, via Roma 56, 53100 Siena, Italy.
agnetis@diism.unisi.it
[3] DIGEP, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy, CNR, IEIIT, Torino, Italy.
federico.dellacroce_a@polito.it

Keywords: Scheduling, bilevel optimization.

1 Introduction

In this paper we focus on a class of scheduling problems where two agents are concerned by the scheduling of a set of n jobs. The first agent, called the *leader*, first select jobs that are given to the second agent, called the *follower*, who next schedules them. This setting falls into the category of *bilevel optimization* [Dempe et al., 2015]. In such problems it is assumed that the leader and the follower follow their own objectives which can be contradictory, so leading to very hard optimization problems. Recently, many papers on bilevel combinatorial optimization have been published (e.g., [Caprara et al., 2016], [Della Croce and Scatamacchia, 2020], [Fischetti et al., 2017], [Fischetti et al., 2018], [Fischetti et al., 2019], [Woeginger, 2021]). On the other hand, to the authors knowledge, the literature on bilevel scheduling is much more limited ([Abass, 2005], [Karlof and Wang, 1996], [Kis and Kovacs, 2012], [T'kindt et al., 2024b]). This paper provides new results with respect to [T'kindt et al., 2024a].

It is assumed that a set N jobs is given to the leader to be scheduled on a single disjunctive machine. Each job j is defined by a processing time p_j, a due date d_j and, depending on the problem, a weight w_j^L. With respect to a given schedule, we denote by C_j the completion time of job j. The leader has to select $n \leq N$ jobs, with n fixed, so as to minimize his objective function $f^L \in \{\sum_j C_j^L, \sum_j w_j^L C_j^L, L_{max}^L, \sum_j T_j^L, \sum_j w_j^L T_j^L, \sum_j U_j^L, \sum_j w_j^L U_j^L, \sum_j w_j^L V_j^L\}$. These jobs are then given to the follower who schedules them to minimize the maximum lateness L_{max}^F. To be able to unambiguously define the notion of optimality we add the hypothesis that the follower, among all his optimal solutions, returns the one that minimizes the leader's objective function (optimistic sce-

nario).

The above objective functions are classic ones as defined in any textbook on scheduling (e.g. [Pinedo, 2022]). The only particular objective function $\sum_j w_j^L V_j$ is defined as the weighted number of early jobs with $V_j = 1$ iff $C_j \leq d_j$; 0 otherwise. It has to be maximized. Notice that in traditional single-level scheduling on a single machine, maximizing $\sum_j w_j^L V_j$ is equivalent to minimizing $\sum_j w_j^L U_j$: this result does no longer holds in a bilevel setting when they are optimized by the leader.

Property 1. For any regular objective function f^F, solving problem $1|OPT - n|\sum_j w_j V_j^L, f^F$ is not equivalent to solving problem $1|OPT - n|\sum_j w_j U_j^L, f^F$.

Proof. The proof is done by exhibiting a generic counter example. Consider the following instance with $N = 4$, $n = 2$ and the follower minimizes f^F.

j	p_j	d_j	w_j
1	6	5	100
2	11	23	40
3	12	23	40
4	14	24	100

If the leader maximizes $\sum_j w_j V_j^L$ then its optimal solution is reached when selecting $\{1, 4\}$: whatever the permutation computed by the follower, we have $\sum_j w_j V_j^L = 100$, which is the maximum value achievable among all possible selections, and $\sum_j w_j U_j^L = 100$. Now, if the leader minimizes $\sum_j w_j U_j^L$, the best selection is $\{2, 3\}$: whatever the permutation computed by the follower, we have $\sum_j w_j U_j^L = 0$ and $\sum_j w_j V_j^L = 80$. □

2 Contributions

We focus on the complexity status of 8 single machine scheduling problems for which the follower minimizes the maximum lateness L_{max}^F. Following the notation of scheduling problems extended in [T'kindt et al., 2024b], these problems are denoted by $1|OPT - n|f^L, L_{max}^F$. When the leader has taken his decisions, i.e. selected n jobs, the follower's problems reduces to the lexicographical optimization problem $1||Lex(L_{max}^F, f^L)$: this highlights interesting links between multiobjective scheduling and bilevel optimistic scheduling. Another important remark is that, when dealing with bilevel optimization, it may happen that some problems are hard with respect to the second level of the Polynomial Hierarchy (e.g. [Woeginger, 2021]): in this case, we say they are Σ_2^p-hard. Consequently, studying the complexity of the 8 scheduling problems $1|OPT - n|f^L, L_{max}^F$ implies considering whether they can be solved in polynomial time, are $\mathcal{NP}$-hard or σ_2^p-hard.

A generic way to model bilevel optimization problems is to use bilevel mathematical programming and, in general, solving a bilevel program is Σ_2^p-hard.

However, it turns out that some bilevel optimization problems may be also formulated by means of (single level) integer linear programming with a polynomial number of variables and constraints. [Woeginger, 2021] recalls that, unless $\mathcal{NP} = \Sigma_2^p$, these problems cannot be Σ_2^p-hard.
For the 8 bilevel scheduling problems considered in this paper, we show the existence of a generic integer linear programming model which rules out the possibility that they are Σ_2^p-hard. This model will be presented at the conference. The obtained complexity results are given in Table 1.

Problem	**Complexity**
$1\|OPT - n\|\sum_j C_j^L, L_{max}^F$	not Σ_2^p-hard
$1\|OPT - n\|\sum_j w_j^L C_j^L, L_{max}^F$	strongly $\mathcal{NP}$-hard
$1\|OPT - n\|\sum_j U_j^L, L_{max}^F$	strongly $\mathcal{NP}$-hard
$1\|OPT - n\|\sum_j w_j^L U_j^L, L_{max}^F$	strongly $\mathcal{NP}$-hard
$1\|OPT - n\|\sum_j w_j^L V_j^L, L_{max}^F$	strongly $\mathcal{NP}$-hard
$1\|OPT - n\|\sum_j T_j^L, L_{max}^F$	at least weakly $\mathcal{NP}$-hard
$1\|OPT - n\|\sum_j w_j^L T_j^L, L_{max}^F$	strongly $\mathcal{NP}$-hard

Table 1. Complexity results when the follower minimizes L_{max}^F

Most of the results presented in Table 1 are derived from the complexity of the follower's problems, i.e. the lexicographical problems. Problem $1|OPT - n|\sum_j C_j^L, L_{max}^F$ is intriguing as, with respect to class $\mathcal{NP}$, its complexity status remains open. Particular cases, as shown at the conference, are polynomially solvable but if this is still true in the general case.

3 Conclusions and perspectives

Bilevel scheduling is an emerging research field and can be seen as an extension of classical multiobjective scheduling and multiagent scheduling. Decisions are made in a hierarchical way which implies to rethink how the problems can be solved. It also asks many new research questions as, to illustrate, how to design efficient upper and lower bounds or even exact algorithms.

From a complexity point of view, it is also interesting to study when a bilevel optimistic scheduling problem turns to be Σ_2^p-hard: the problems considered in this paper remain, at most, hard with respect to $\mathcal{NP}$ but what happens if we complexify them by adding constraints or considering more complex shop environments?

References

[Abass, 2005] Abass, S. (2005). Bilevel programming approach applied to the flow shop scheduling problem under fuzziness. *Computational Management Science*, 2:279–293.

[Caprara et al., 2016] Caprara, A., Carvalho, M., Lodi, A., and Woeginger, G. (2016). Bilevel knapsack with interdiction constraints. *INFORMS Journal on Computing*, 28:319–333.

[Della Croce and Scatamacchia, 2020] Della Croce, F. and Scatamacchia, R. (2020). An exact approach for the bilevel knapsack problem with interdiction constraints and extensions. *Mathematical Programming*, 183:249–281.

[Dempe et al., 2015] Dempe, S., Kalashnikov, V., Pérez-Valdés, G., and Kalashnikova, V. (2015). *Bilevel Programming Problems: Theory, Algorithms and Applications to Energy Networks*. Springer.

[Fischetti et al., 2017] Fischetti, M., Ljubic, I., Monaci, M., and Sinnl, M. (2017). A new general-prupose algorithm for mixed-integer bilevel linear programs. *Operations Research*, 65:1615–1637.

[Fischetti et al., 2018] Fischetti, M., Ljubic, I., Monaci, M., and Sinnl, M. (2018). On the use of intersection cuts for bilevel optimization. *Mathematical Programming*, 172:77–103.

[Fischetti et al., 2019] Fischetti, M., Ljubic, I., Monaci, M., and Sinnl, M. (2019). Interdiction games and monotonicity with application to knapsack problems. *INFORMS Journal on Computing*, 31:390–410.

[Karlof and Wang, 1996] Karlof, J. and Wang, W. (1996). Bilevel programming applied to the flow shop scheduling problem. *Computers & Operations Research*, 23(5):443–451.

[Kis and Kovacs, 2012] Kis, T. and Kovacs, A. (2012). On bilevel machine scheduling problems. *OR Spectrum*, 34:43–68.

[Pinedo, 2022] Pinedo, M. (2022). *Scheduling: Theory, Algorithms and Systems*. Springer.

[T'kindt et al., 2024a] T'kindt, V., Della Croce, F., and Agnetis, A. (2024a). Bilevel adversarial single machine scheduling problems with job selection. *19th International Workshop on Project Management and Scheduling (PMS'24), Bern, Switzerland*.

[T'kindt et al., 2024b] T'kindt, V., Della Croce, F., and Agnetis, A. (2024b). Single machine adversarial bilevel scheduling problems. *European Journal of Operational Research*, 315(1):63–72.

[Woeginger, 2021] Woeginger, G. (2021). The trouble with the second quantifier. *4OR*, 19:157–181.

A new formulation for parallel machine scheduling with conflicts

Phablo F. S. Moura, Roel Leus, and Hande Yaman

Research Center for Operations Research & Statistics, KU Leuven, Belgium
{phablo.moura, roel.leus, hande.yaman}@kuleuven.be

Keywords: integer programming, scheduling, valid inequalities, branch and cut.

1 Introduction & literature

Given an undirected graph $G = (V, E)$ (a.k.a. *conflict graph*) where V is a set of n vertices (representing the jobs), processing times $p\colon V \to \mathbb{Z}_>$, and $m \geq 2$ identical machines, the problem Parallel Machine Scheduling with Conflicts (PMC) consists in finding an assignment $c\colon V \to \{1, \ldots, m\}$ with $c(u) \neq c(v)$ for all $\{u, v\} \in E$ that minimizes $\max_{k\in\{1,\ldots,m\}} \sum_{v\in V\colon c(v)=k} p(v)$, that is, the makespan. This problem is clearly NP-hard as it contains both 3-PARTITION and GRAPH m-COLORABILITY (Garey and Johnson 1979).

Let $[\ell]$ denote the set $\{1, \ldots, \ell\}$ for any $\ell \in \mathbb{Z}_>$. The following linear formulation for PMC, subsequently referred to as *assignment formulation* (AF), was introduced by Kowalczyk and Leus (2017). For each $v \in V$ and $k \in [m]$, there is a binary variable x_{vk} indicating the machine that processes the job: $x_{vk} = 1$ if and only if job v is assigned to machine k. Moreover, the model contains a non-negative real variable y whose value is at least the maximum processing time over all machines $k \in [m]$.

$$
\begin{aligned}
\text{(AF)} \qquad & \min\ y \\
& \text{s.t.} \sum_{k\in[m]} x_{vk} \geq 1, && \forall v \in V, && (1)\\
& x_{uk} + x_{vk} \leq 1 && \forall \{u, v\} \in E, k \in [m], && (2)\\
& \sum_{v\in V} p(v) x_{vk} \leq y && \forall k \in [m], && (3)\\
& x_{vk} \in \{0, 1\} && \forall v \in V, k \in [m], && (4)\\
& y \geq 0. && && (5)
\end{aligned}
$$

Constraints (1) guarantee that each job is assigned to at least one machine, while constraints (2) imply that no conflicting jobs are assigned to the same machine. The objective function together with constraints (3) establish the minimum makespan.

This problem was first studied by Bodlaender et al. (1994), who designed polynomial-time approximation schemes (PTASs) for the cases where the conflict graph is bipartite, complete multipartite, or has bounded treewidth. More recently, Furmańczyk et al. (2024) showed a 2-approximation algorithm for PMC, and a PTAS for the case when the jobs have processing times of unit duration. Problem PMC restricted to unit-time jobs is NP-hard even on bipartite and interval graphs (Mallek and Boudhar 2024). A number of solving methods based on mixed-integer linear programming (MILP) have also been proposed for PMC in the literature. One is the branch-and-price algorithm due to Bianchessi and Tresoldi (2021), which uses a MILP formulation with a binary variable for each stable set of the conflict graph and each machine, and a non-negative real variable for every machine. Another algorithm (the first-published computational study, to our knowledge) is a binary search using a set-covering model for Bin Packing with Conflicts (BPPC) devised by Kowalczyk

and Leus (2017). Given $m \in \mathbb{Z}_>$ identical bins with capacity $C \in \mathbb{Z}_>$, a set V of n items with capacity consumption $p\colon V \to \mathbb{Z}_>$, and a (conflict) graph $G = (V, E)$, BPPC consists in finding an assignment of items to a minimum number of bins so that the capacity of the bins is respected and no pair of items in E are assigned to the same bin. Note that the decision versions of PMC and BPPC are equivalent: an instance (G, p, m) of PMC has a solution of makespan C if and only if the items in $V(G)$ can be assigned (without conflicts) to m bins of capacity C. Sadykov and Vanderbeck (2013) proposed a branch-and-price approach to BPPC, and Epstein and Levin (2008) designed constant-factor approximation algorithms for the problem restricted to bipartite graphs and perfect graphs. Mallek et al. (2019) proposed heuristics and MILP formulations for a variant of PMC where the machines run at different speeds and all jobs are unit time. This problem was shown to be NP-hard even when restricted to instances with two machines and a forest as the conflict graph (Mallek and Boudhar 2024).

2 Link with other combinatorial problems

Scheduling with conflicts is closely related to the vertex coloring problem (VCP), which consists in, given a graph $G = (V, E)$, finding a coloring $c\colon V \to [\ell]$ with $\ell \in \mathbb{Z}_>$ colors such that $c(u) \neq c(v)$ for all $\{u, v\} \in E$, and ℓ is minimum. The smallest $\ell \in \mathbb{Z}_>$ such that G admits an ℓ-coloring is denoted by $\chi(G)$. Obviously, any feasible solution for an instance of PMC with conflict graph G and m machines is a (proper) vertex coloring of G with m colors. Thus an instance (G, p, m) of this problem is infeasible if and only if $m < \chi(G)$. Problem VCP is a classical optimization problem with a vast literature devoted to algorithms, complexity, structural aspects, and computational experiments (see, e.g., Malaguti and Toth (2010), and Tuza (1997)). It is therefore natural to use the knowledge about VCP to study PMC. In this work, we focus on MILP formulations, strong valid inequalities, and separation algorithms.

We observe that PMC is also related to so-called *interval scheduling*. In the basic interval scheduling problem, given a set of job intervals $[s_i, f_i)$ for all $i \in [n]$ (with s_i and f_i the start and end time of job i), the objective is to assign jobs to machines so that no two job intervals assigned to the same machine overlap, while minimizing the number of machines used. The input can be viewed as a conflict graph with one vertex for each interval, and an edge $\{u, v\}$ whenever the intervals corresponding to u and v overlap. Hence, the basic interval scheduling problem corresponds exactly to the vertex coloring problem on interval graphs, which is in turn a particular case of BPPC. For more results on interval scheduling, the interested reader is referred to the survey by Kolen et al. (2007).

3 Contributions

In Moura et al. (2025), we consider the natural assignment formulation (AF) for PMC using binary variables indexed by the jobs and machines, and discuss how to reduce symmetries in this model. Note that the optimal value of the linear relaxation of (AF) is precisely $\frac{1}{m}\sum_{v \in V} p(v)$; the average makespan is a trivial lower bound for PMC. We also propose a compact MILP formulation for PMC to alleviate some issues related to symmetry and unbalancedness of the enumeration tree associated with (AF). The proposed formulation for PMC uses a set of representative jobs (one for each machine) to represent feasible solutions to the problem, and is based on the asymmetric representatives model for VCP introduced by Campêlo et al. (2008). A similar idea of representatives is used to model the classical one-dimensional bin packing problem in Hadj Salem and Kieffer (2020). Consider an instance (G, p, m) of PMC with $G = (V, E)$. The complement of G, denoted by $\bar{G}$, is the graph

$(V, \bar{E})$ where $\bar{E} = \{\{u, v\} \subset V : \{u, v\} \notin E, u \neq v\}$ is the complement of E. We denote by $\bar{e}$ the size of $\bar{E}$. Let $\prec$ be an arbitrary ordering on V. We remark that this ordering is not related to the order in which the jobs are processed (the latter ordering is irrelevant for the makespan objective). Note also that the ordering $\prec$ avoids symmetrical solutions and affects the dimension of polytope associated with the representatives formulation of the problem. Before showing the model, we present some additional useful notation. Let $\bar{N}^-(v) = \{u \in V : u \prec v, \{u, v\} \notin E\}$ and $\bar{N}^+(v) = \{u \in V : v \prec u, \{u, v\} \notin E\}$ be the negative and positive anti-neighborhood of v, respectively. Define $\bar{N}[v] = \bar{N}^+(v) \cup \bar{N}^-(v) \cup \{v\}$, $\bar{N}^-[v] = \bar{N}^-(v) \cup \{v\}$, and $\bar{N}^+[v] = \bar{N}^+(v) \cup \{v\}$.

We next describe the representatives formulation (RF) for PMC on input (G, p, m) with $G = (V, E)$. For each $v \in V$ and $u \in \bar{N}^-[v]$, define a variable $\tilde{x}_{uv} \in \{0, 1\}$ that equals 1 if and only if the machine represented by job u contains job v. Additionally, there is a real variable y that is an upper bound for the processing time of each machine.

$$
\begin{aligned}
\text{(RF)} \qquad & \min\ y && && (6)\\
& \text{s.t.} \sum_{v \in V} \tilde{x}_{vv} \leq m && && (7)\\
& \sum_{u \in \bar{N}^-[v]} \tilde{x}_{uv} \geq 1 && \forall v \in V, && (8)\\
& \tilde{x}_{vu} + \tilde{x}_{vw} \leq \tilde{x}_{vv} && \forall v \in V, u, w \in \bar{N}^+(v) \text{ with } \{u, w\} \in E, && (9)\\
& \tilde{x}_{vu} \leq \tilde{x}_{vv} && \forall v \in V, u \in \bar{N}^+(v), && (10)\\
& \sum_{u \in \bar{N}^+[v]} p(u)\tilde{x}_{vu} \leq y && \forall v \in V, && (11)\\
& \tilde{x}_{uv} \in \{0, 1\} && \forall v \in V, u \in \bar{N}^-[v], && (12)\\
& y \geq 0. && && (13)
\end{aligned}
$$

The objective function (6) together with constraints (11) minimizes the maximum processing time of the machines. A solution that uses at most m machines is guaranteed by constraint (7). Constraints (8) ensure that each job v is assigned to one machine, considering machines represented by v or by any of its negative anti-neighbors in G. Constraints (9) guarantee that no conflicting jobs are assigned to the same machine. Moreover, if a job is not a representative, it cannot represent any job according to constraints (10).

In Moura et al. (2025) we conduct a polyhedral study of the polytope associated with (RF) and establish some classes of valid inequalities inherited from the stable set polytope that are induced by specific classes of subgraphs. We also briefly discuss the separation problems of some of these classes of inequalities. Our proposed solution method based on the (compact) representatives formulation is easier to implement than the best algorithms currently known for PMC, which are all based on models with exponentially many variables and NP-hard pricing problems. Furthermore, the proposed method for PMC is more amenable to enhancements, such as the inclusion of new cuts derived for the stable set and vertex coloring problems.

4 Results

Computational experiments on the hardest instances in the benchmark instance set for PMC show that the proposed algorithms are in general superior (either in running time or quality of the solutions) to the current state-of-the-art methods. Considering the entire set of instances in the benchmark, the branch-and-cut approach produces essentially the same average optimality gaps as obtained by the other algorithms in the literature. Looking

only at the instances with positive gap, our average gaps are up to 10 times smaller for the instances with a graphs density of at least 0.3. We also observe from our computational results that most of the (feasible) solved instances in the benchmark dataset with at least 50 vertices have their optimal value very close to (at most 5% larger than) the trivial lower bound $\lceil \sum_{v \in V} p(v)/m \rceil$ for PMC. This implies that the conflicts in these instances do not really affect the optimal value very much compared to the optimal makespan when scheduling the same jobs without conflicts. To better understand the impact of the conflicts, we construct extra instances where the difference between the optimal value and the trivial lower bound can be large, and present empirical evidence that our method is more likely to outperform the best algorithm in the literature as this difference increases.

References

Bianchessi N. and E. Tresoldi. A stand-alone branch-and-price algorithm for identical parallel machine scheduling with conflicts. *Computers & Operations Research* 136:105464, 2021.

Bodlaender H.L., K. Jansen, and G. J. Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics* 55(3):219–232, 1994.

Campêlo M., V. Campos, and R. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics* 156(7):1097–1111, 2008.

Epstein L. and A. Levin. On bin packing with conflicts. *SIAM Journal on Optimization* 19(3):1270–1298, 2008.

Furmańczyk H., T. Pikies, I. Sokolowska, and K. Turowski. Approximation algorithms for job scheduling with block-type conflict graphs. *Computers & Operations Research* 166:106606, 2024.

Garey M.R. and D.S. Johnson. *Computers and Intractability*. W. H. Freeman, New York, 1979.

Hadj Salem K. and Y. Kieffer. New symmetry-less ILP formulation for the classical one dimensional bin-packing problem. *In*: D. Kim, R. N. Uma, Z. Cai, and D. H. Lee, editors, Computing and Combinatorics, pages 423–434, Cham, 2020. Springer International Publishing.

Kolen A.W.J., J.K. Lenstra, C.H. Papadimitriou and F.C.R. Spieksma. Interval scheduling: A survey. *Naval Research Logistics* 54(5):530–543, 2007.

Kowalczyk D. and R. Leus. An exact algorithm for parallel machine scheduling with conflicts. *Journal of Scheduling* 20(4):355–372, 2017.

Malaguti E. and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research* 17(1):1–34, 2010.

Mallek A., M. Bendraouche, and M. Boudhar. Scheduling identical jobs on uniform machines with a conflict graph. *Computers & Operations Research* 111:357–366, 2019.

Mallek A. and M. Boudhar. Scheduling on uniform machines with a conflict graph: Complexity and resolution. *International Transactions in Operational Research* 31(2):863–888, 2024.

Moura P.F.S., R. Leus, and H. Yaman. Compact formulations and valid inequalities for parallel machine scheduling with conflicts. *European Journal of Operational Research* 325(3):433–443, 2025.

Sadykov R. and F. Vanderbeck. Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing* 25(2):244–255, 2013.

Tuza Z. Graph colorings with local constraints – A survey. *Discussiones Mathematicae – Graph Theory* 17(2):161–228, 1997.

F - Project scheduling

Project Tightness (PT): A Complementary Topological Indicator for Explaining Project Schedule Behaviour

Fernando Acebes [1], Liangyan Tao[2], Manuel Sobrino[1] and Javier Pajares[1]

[1] GIR INSISOC. School of Industrial Engineering, University of Valladolid, Spain
fernando.acebes, manuel.sobrino, javier.pajares@uva.es

[2]College of Economics and Management, Nanjing Univ. of Aeronautics and Astronautics, China.
lytao@nuaa.edu.cn

Keywords: Project Tightness, Series-Parallel Index, Network Topology, Project Control.

1. Introduction

Topological indicators play a central role in the analysis of project scheduling and control, as they describe the structure of activity networks and help explain why projects with similar characteristics may exhibit different dynamic behaviour. They quantify features such as network size (Vanhoucke et al., 2008, 2016).

Among them, the Series–Parallel Index (SP), initially introduced by Tavares (1999), has become the reference measure for distinguishing between predominantly serial and parallel project structures. Over the last two decades, SP and related measures have been widely used in simulation studies and empirical analyses (Vaseghi et al., 2024; Song and Vanhoucke, 2025; Song et al., 2026), consistently confirming that network topology affects project control performance, particularly in terms of time and cost predictability.

Nevertheless, empirical results from our own simulations indicate that seriality alone does not fully explain project behaviour. Networks with identical SP values may exhibit markedly different responses depending on the distribution of activity slack, revealing a latent structural factor: network tightness, that is, how "stretched" or "tense" project paths are in terms of duration proximity.

Consequently, we propose a new topological indicator, Project Tightness (PT), designed to capture this structural tension and complement the information provided by SP. Jointly considering PT and SP yields a richer and more consistent explanation of schedule sensitivity and control performance, extending topology-based approaches through the integration of structural and temporal dimensions.

2. Project Tightness: A New Topological Indicator

Although the Series–Parallel Index (SP) is a valuable structural descriptor, empirical analyses show that it is not an absolute indicator of project behaviour. Projects with identical SP values may respond differently to uncertainty depending on the proximity in duration of alternative paths, that is, on schedule tightness. When several paths compete for criticality, the network becomes more fragile to minor perturbations even if SP remains unchanged. To capture this dimension, we introduce the Project Tightness (PT) indicator, which quantifies the overall temporal tension of the schedule (Acebes & Pajares, 2025).

Unlike measures such as XSLACK-R (Patterson, 1976), PT accounts not only for the amount of total slack but also for its organisation relative to the dominant critical path, allowing networks with similar average slack but different structural resilience to be distinguished. PT is defined as:

$$PT = 1 - \frac{\sum TF_i}{(n - r) \cdot TPT}$$

where n is the number of activities, r is the number of activities on the longest critical path, TPT is the planned project duration, and TF_i is the total float of activity i.

PT takes values in [0, 1], with higher values indicating tighter schedules. It is bounded, invariant to time-scale changes, and monotonic with respect to parallel criticality. While SP captures structural shape, PT reflects temporal tension; together, they provide a two-dimensional description of project morphology.

Figure 1 illustrates two networks with identical topology and SP = 0.33 but markedly different PT values (0.349 and 0.984), reflecting different degrees of schedule compression and flexibility.

This example highlights how PT captures variations in temporal tightness that remain invisible to SP alone.

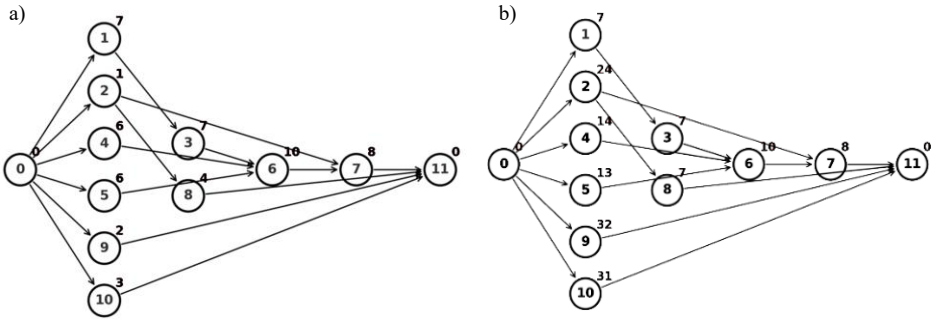


Figure 1. Two project networks with the same Series–Parallel Index (SP = 0.33) but different Project Tightness values (PT_a = 0.349 and PT_b = 0.984), showing how PT reflects differences in schedule tension

3. Case Studies

In this paper, two case studies are presented to illustrate the relevance and practical implications of the proposed Project Tightness (PT) indicator. These experiments aim to demonstrate how PT significantly influences key project outcomes, providing valuable insights into project behaviour under uncertainty. Furthermore, PT complements the traditional Serial–Parallel (SP) indicator: while SP captures the structural arrangement of activities, from more serial to more parallel configurations, PT reveals the underlying temporal tension that determines the project's sensitivity and robustness.

3.1. Correlation analysis between project activities

To demonstrate the importance of the Project Tightness (PT) indicator in explaining project responses under uncertainty, a first case study was conducted, analysing how key project parameters (mean value, variance, and sensitivity indicators (Criticality Index - CI, Cruciality Index - CrI, Schedule Sensitivity Index - SSI)) vary when correlations between two project activities are introduced.

Within the different simulation scenarios, correlations were established in two directions: highly positive and highly negative correlations between activities. Moreover, the correlated activities were selected either from the same initial critical path or from different paths with distinct initial criticalities. Correlations were introduced by imposing fixed Pearson coefficients ($\rho = \pm 0.8$) between the stochastic durations of selected activity pairs using a Gaussian copula, while preserving marginal duration distributions.

The analysis was conducted using a set of fictitious project networks, whose Serial–Parallel (SP) indicator ranged from approximately 0.1 (highly parallel projects) to 0.9 (highly serial projects). For each network, the PT indicator was calculated based on its structural and temporal properties. In addition, for the same projects (while maintaining their SP value), we modified the duration of specific activities to generate tighter configurations by proportionally reducing the total floats of non-critical activities, while preserving the precedence structure and thus maintaining a constant SP value. One of the main results is shown in Figure 2, which depicts the relationship between PT and the percentage change in project variance when activity correlations are introduced ($\rho = 0.8$), distinguishing between networks with high and low SP structures. To distinguish SP regimes, networks were split by the sample median SP (SP ≥ median: High-SP; otherwise: Low-SP).

A general downward trend can be observed: as PT increases, the variance sensitivity decreases, indicating a lower capacity for correlation propagation in tighter networks. However, the steeper slope observed for low-SP projects suggests that more parallel networks are more sensitive to internal correlation effects when they exhibit moderate tension levels. Overall, the figure confirms that PT moderates the effect of SP on project variance, revealing a differentiated behaviour between networks with similar structural arrangements but distinct levels of temporal tightness.

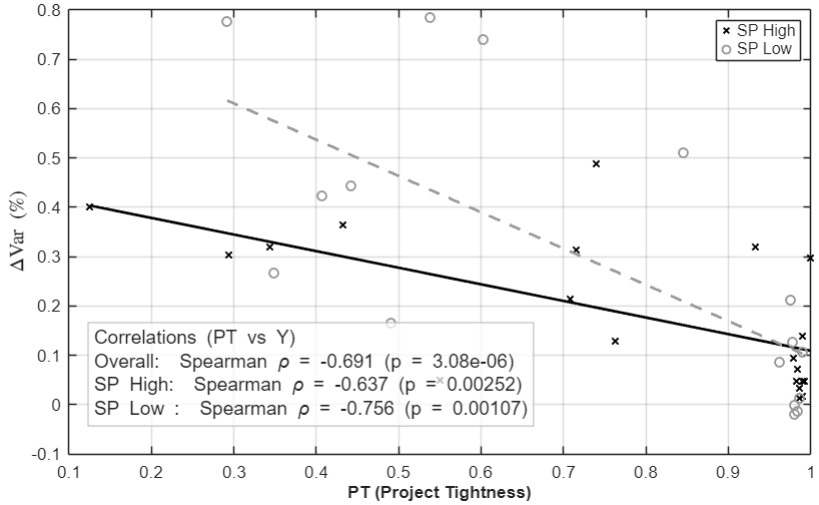


Figure 2. Relationship between Project Tightness (PT) and project variance, showing distinct trends for high and low Serial–Parallel (SP) structures

3.2. Time-control simulations

The second simulation study was inspired by the experimental framework proposed by Vanhoucke (2010). Two fictitious project networks were selected, representing opposite Serial–Parallel (SP) configurations (one highly parallel (SP ≈ 0.2) and one highly serial (SP ≈ 0.8)). For both projects, the Total Contribution (TC) metric was computed following Vanhoucke's corrective-action simulation approach. To examine the additional influence of Project Tightness (PT), each network was also modified to create a "tighter" version, with PT values close to 1, representing schedules with minimal slack and stronger temporal tension.

The results, presented in Figure 3, indicate that when PT remains low, the outcomes are consistent with those reported by Vanhoucke (2010): low-SP projects (i.e., more parallel networks) achieve greater total contributions under corrective actions, whereas high-SP networks show more limited improvements. However, as PT increases, this pattern changes: tighter projects display larger deviations in TC, and in some cases, the expected relationship between SP and TC is even reversed.

These findings suggest that PT complements the traditional SP indicator, offering an additional dimension to explain project control performance. By integrating both structural (SP) and temporal (PT) characteristics, the analysis provides a more comprehensive understanding of how network topology and schedule tension jointly influence the effectiveness of corrective actions.

4. Conclusions

This paper introduced Project Tightness (PT) as a topological indicator that captures the temporal tension of project networks and complements the traditional Series–Parallel (SP) index. Through two simulation studies, focused on variance propagation and corrective-action performance, we demonstrated that considering PT alongside SP provides a more complete and consistent understanding of project behaviour.

While SP describes the structural configuration of activities, PT reflects the degree of schedule compression and its impact on project sensitivity and robustness. The first study showed that networks with comparable SP values may exhibit distinct variance responses depending on their internal tension. The second confirmed that Total Contribution (TC) under corrective actions depends not only on SP, as reported in earlier work, but also on PT, since tighter projects can amplify or invert expected patterns of control performance. Overall, the joint use of SP and PT provides a practical framework for classifying and interpreting project behaviour under uncertainty, linking structural and temporal perspectives. By combining structural and temporal dimensions, it helps explain differences in performance across networks and supports more informed control decisions.

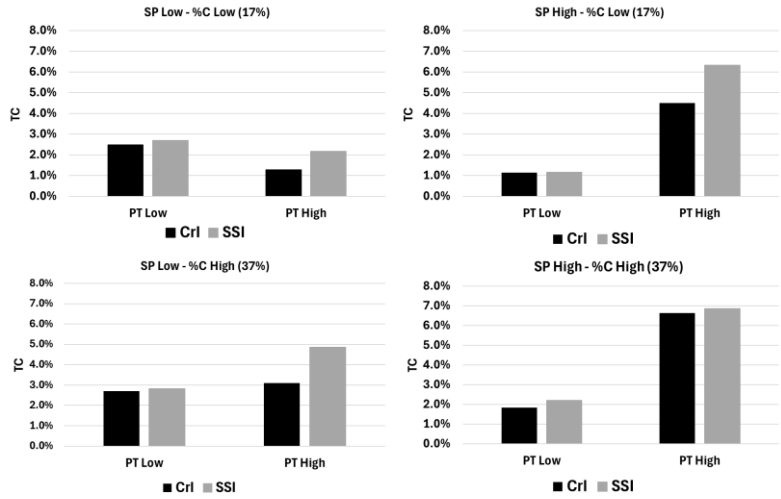


Figure 3. Impact of Project Tightness (PT) on Total Contribution (TC) for Project Networks with Different Serial–Parallel (SP) Structures, Extending the Topology-Based Analysis of Vanhoucke (2010)

Future research could revisit previous studies based solely on the SP indicator to examine whether their conclusions remain consistent when project tension (PT) is considered. In addition, this approach could be extended to resource-constrained environments, where schedule tightness and structural configuration may interact differently, further testing the robustness and practical relevance of the PT indicator.

Funding

This project has been partially funded by the Regional Government of Castile and Leon (Spain) under the Regional Accredited Research Groups Support Program, with grant VA042G2.

References

Acebes, F., & Pajares, J. (2025). Project Tightness Index (PT): A Concise Metric of Schedule Tightness. In *GIR INSISOC*. https://doi.org/10.13140/RG.2.2.13545.15200

Patterson, J. H. (1976). Project scheduling: The effects of problem structure on heuristic performance. *Naval Research Logistics Quarterly, John Wiley & Sons*, *23(1)*, 95–123. https://doi.org/10.1002/nav.3800230110

Song, J., Song, J., Browning, T., & Vanhoucke, M. (2026). Project monitoring and control with an empirically grounded budget-release model. *European Journal of Operational Research*, *328*(2), 646–667. https://doi.org/10.1016/j.ejor.2025.06.007

Song, Y., & Vanhoucke, M. (2025). Schedule risk analysis for project control with risk interactions. *Annals of Operations Research*. https://doi.org/10.1007/s10479-025-06668-8

Tavares, L. V. (1999). Advanced Models for Project Management. In *Advanced Models for Project Management*. Springer US. https://doi.org/10.1007/978-1-4419-8626-9

Vanhoucke, M. (2010). Using activity sensitivity and network topology information to monitor project time performance. *Omega*, *38*(5), 359–370. https://doi.org/https://doi.org/10.1016/j.omega.2009.10.001

Vanhoucke, M., Coelho, J., & Batselier, J. (2016). An overview of project data for integrated project management and control. *The Journal of Modern Project Management*, *3*(2), 6–21. http://www.projectmanagement. ugent.be/research/data

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., & Tavares, L. V. (2008). An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, *187*(2), 511–524. https://doi.org/10.1016/j.ejor.2007.03.032

Vaseghi, F., Martens, A., & Vanhoucke, M. (2024). Analysis of the impact of corrective actions for stochastic project networks. *European Journal of Operational Research*, *316*(2), 503–518. https://doi.org/10.1016/j.ejor.2024.02.040

The Role of Correlations Between Activity Durations in Project Schedule Risk Analysis

Javier Pajares[1], Fernando Acebes[1], Natalia Martín-Cruz[1] and Ricardo Gúdel[1]

[1] GIR INSISOC – University of Valladolid, SPAIN
javier.pajares, fernando.acebes, ambiela, ricardo.gudel@uva.es

Keywords: Schedule Risk analysis, Correlations, Risk Sensitivity Indexes, Project Tightness

1. Introduction

The literature on project management highlights correlations between activity durations in real projects and their influence on project performance. In many cases, the correlations are due to the presence of common risk factors affecting multiple activities simultaneously: adverse weather conditions (Acebes et al., 2014), resource availability or labour productivity (and expertise), design errors and changes in the scope of the project that affect several activities, etc.

Negative correlations also occur. For example, due to the learning effect, i.e., the experience and knowledge gained in one activity can make teams more efficient in other activities requiring similar skills (Edmondson & Nembhard, 2009). Negative correlations may also arise from managerial or resource-driven compensatory decisions, e.g., reactive scheduling (Herroelen & Leus, 2005). Delays in one activity trigger acceleration, resource reallocation, or scope adjustments in another activity, leading to an inverse relationship between their durations. Also, negative correlations are result of decisions to overlap activities or the implementation of fast-tracking, where delays in upstream activities force downstream tasks to reduce their duration through reduced rework or increased parallelization (Krishnan et al., 1997).

CPM and PERT literature usually consider that activity durations are independent (MacCrimmon & Ryavec, 1964; Vanhoucke, 2013). Banerjee & Paul (2008) alert that correlation between activity durations can affect PERT bias.

The assumption of independence was also inherited by much of the Schedule Risk Analysis (SRA) practice. Although simulation would explicitly allow correlation to be introduced, in most industrial and academic applications, the simplified assumption of independence has been retained (Ökmen & Öztaş, 2008; Yang, 2007)

However, correlations between activity durations can affect project uncertainty and its statistical characteristics (Kaut et al., 2021). Van Dorp & Duffey (1999) show that correlations affect the distribution shape of the project total duration, and the changes are higher the higher the size of the project (number of activities). Similar studies have concluded that activity correlation can affect the expected duration of a project and its variance.

In this paper, we go one step further and investigate how correlations can influence risk sensitivity indexes, such as criticality (CI, the probability of the activity to fall on the critical path, cruciality (CrI, the correlation between the duration of an activity and the total project duration) and SSI (critically multiplied by the quotient between the standard deviations of the activity and the project). To this aim, we perform Monte Carlo simulations, comparing results with and without correlations.

After verifying that correlations do indeed change some of the sensitivity indexes, we wonder to what extent different variables in the network topology affect these changes. Specifically, we asked ourselves how the series-parallel (S/P) degree, or the presence of critical paths with similar or different durations, affects these changes. We also studied the influence of the network's degree of tightness, using a new indicator, Project Tightness (PT) (Acebes & Pajares, 2025). PT quantifies the degree of compression or "tension" in the schedule, taking into account both slack in activities and the configuration of the project's critical structure, and normalizes the total slack by the number of activities not on the dominant critical path.

Our results show that correlations have a significant impact on project variance, cruciality, and SSI; S/P is relevant, but PT has a greater impact on correlations' effects on sensitivity indexes.

This research has practical implications for project risk management and monitoring. The objective of the SRA methodology (Hulett, 1996) is to identify which activities contribute most to the project's total variability. To do this, the sensitivity indexes of the different activities are estimated, and threshold values are set, so that monitoring and control efforts are intensified in those

activities that exceed them (Vanhoucke, 2010). Consequently, if correlations change the values of the sensitivity indexes, the activities on which to focus monitoring efforts will differ from those considered when there are no correlations.

2. Simulations and results

A controlled Monte Carlo simulation experiment was conducted on artificially generated project networks to analyse the impact of activity-duration correlations under different topological conditions. Artificial networks were generated using RANGEN, systematically varying their series–parallel degree (SP) and Project Tightness (PT) in order to obtain structurally diverse project configurations. For each SP–PT configuration, a large number of Monte Carlo simulation runs were performed.

Activity durations were modelled as stochastic variables and analysed under four correlation scenarios: positive and negative correlation (±0.8) between two activities located either on the same path or on different paths. These scenarios were compared against a baseline case assuming independent activity durations. Correlation was introduced pairwise to isolate its effect and ensure comparability across scenarios. For each experimental setting, relative changes with respect to the baseline case were computed for project mean duration, variance, and the 90th percentile (P90) of the makespan distribution. In addition, activity risk-sensitivity indexes commonly used in Schedule Risk Analysis—criticality (CI), cruciality (CrI), and the Schedule Sensitivity Index (SSI)—were estimated and analysed comparatively across topological configurations.

In this section, we show the results of the simulations performed using the software MCSimulRisk (Acebes et al., 2023). Artificial project networks with different topologies were generated using RANGEN (Demeulemeester et al., 2003).

In Table 1, the first and second columns refer to the project network topology (SP series/parallel, PT Project Tightness). The following columns show, in percentage, the increments (decrements) in the project mean, variance, and the 90% percentile of the total project duration distribution, when there are correlations in the project (between two activities), compared to the scenario with no correlation. We show scenarios in which correlated activities are on the same (S) or different (P) path. The numbers in brackets are the correlation values (i.e., +0.8 or -0.8).

We see that, regardless of the network topology, the increase in the project mean duration is not very different when correlations are introduced (changes below 1%). However, correlations produce a significant increase (a decrease when negative), especially when they affect activities along the same path (S). For instance, with a correlation of +0.8, SP=0.111, and low PT, the maximum increment in variance is 78.64%; correlations produce a significant change in project uncertainty. However, if activities are on different paths, the increment is only 1.139%.

Furthermore, the variance increment appears to increase with SP, and for the same value of SP, projects with low tightness (PT Low) exhibit a much higher variance increment than those with high tightness.

Table 1. Influence of correlations and net topology on mean, variance, and 90 th percentile

Topological Index		ΔMean (%)				ΔVar (%)				ΔP90 (%)			
SP	PT	S(+0.8)	S(–0.8)	P(+0.8)	P(–0.8)	S(+0.8)	S(–0.8)	P(+0.8)	P(–0.8)	S(+0.8)	S(–0.8)	P(+0.8)	P(–0.8)
0.111	Low	0.149	-0.015	0.113	-0.030	78.460	-77.485	1.139	-0.299	3.620	-4.937	0.186	-0.047
0.111	High	0.331	-0.312	-0.185	0.080	12.701	-4.708	5.334	-9.415	0.712	-0.221	0.101	-0.167
0.333	Low	0.085	-0.042	0.054	-0.070	42.322	-40.281	0.085	-1.013	1.548	-1.544	0.172	-0.121
0.333	High	0.232	-0.228	-0.037	-0.062	51.027	-48.595	10.906	-11.737	1.774	-1.990	0.015	-0.108
0.556	Low	0.075	0.004	0.053	-0.042	48.698	-48.800	1.263	-2.298	1.717	-1.793	0.123	-0.115
0.556	High	0.151	-0.091	-0.205	0.198	13.908	-7.492	8.246	-13.434	0.504	-0.172	0.035	-0.055
0.778	Low	0.016	-0.036	0.034	-0.058	31.847	-27.282	1.074	-0.820	0.840	-0.815	0.153	-0.027
0.778	High	0.205	-0.117	-0.250	0.111	7.178	-8.031	10.803	-14.831	0.377	-0.260	-0.134	0.015
0.889	Low	0.010	0.008	0.026	-0.012	40.071	-42.553	-2.057	-0.591	1.108	-1.221	-0.036	-0.002
0.889	High	0.146	-0.033	0.027	0.027	31.994	-32.540	14.090	-11.774	0.816	-0.846	0.135	-0.020

Table 2 summarises the same experimental scenarios and shows the variations in criticality (CI), cruciality (CrI), and SSI. Changes in CI remain small across all cases, with maximum deviations

below six points, confirming that correlations have a limited effect on the probability of activities becoming critical.

Table 2. Influence of correlations and net topology on criticality, cruciality and SSI.

Topological Index		ΔCI				ΔCrI				ΔSSI			
SP	PT	S(+0.8)	S(–0.8)	P(+0.8)	P(–0.8)	S(+0.8)	S(–0.8)	P(+0.8)	P(–0.8)	S(+0.8)	S(–0.8)	P(+0.8)	P(–0.8)
0.111	Low	-1.012	-0.656	0.252	0.536	57.233	-60.412	59.723	58.287	-19.614	84.831	0.595	-0.490
0.111	High	4.744	-5.784	-4.308	3.136	33.294	-15.641	21.952	-14.061	4.301	-5.327	-4.774	5.003
0.333	Low	0.340	-0.076	-0.396	0.428	42.237	-39.811	57.098	55.724	-11.640	22.399	0.808	-0.808
0.333	High	-2.924	-3.764	-3.332	2.468	30.376	-13.854	38.336	26.421	-5.356	6.372	-5.640	3.976
0.556	Low	0.004	0.000	0.000	0.000	43.901	-36.830	43.447	43.411	-14.119	31.510	0.802	1.082
0.556	High	3.512	-4.140	-5.268	-3.940	33.183	-13.460	43.960	-24.811	-2.542	2.804	-6.257	7.254
0.778	Low	0.000	0.000	0.000	0.000	40.070	40.897	44.343	43.979	-7.172	9.915	0.807	-0.642
0.778	High	1.180	0.688	5.548	-4.688	15.315	-18.119	49.108	17.081	-2.507	1.949	-5.063	6.842
0.889	Low	0.000	0.000	0.000	0.000	28.166	-40.413	41.668	41.686	-8.312	17.165	0.642	0.562
0.889	High	-0.752	0.504	-1.644	-1.460	25.806	-34.314	52.995	25.295	-5.874	9.785	-6.359	6.969

By contrast, CrI exhibits substantial variations in every scenario. The most pronounced positive and negative shifts appear consistently in networks with low PT, where ΔCrI values frequently exceed ±55 points. SP acts as a secondary amplifying factor: within low-PT configurations, the largest shifts occur when SP is also low. However, the comparison across scenarios shows that PT is the dominant factor moderating the sensitivity of CrI to correlations, as high-PT networks exhibit much smaller changes even under identical SP conditions and correlation patterns.

SSI follows a different behaviour. Its largest increments arise under negative correlations and when the correlated activities belong to the same path, with values exceeding +80 points in some low-SP/low-PT cases. This reflects a stronger transmission and amplification of variability along the dominant project chain when negative dependence affects sequential activities.

Figure 1 provides a graphical summary of the ΔCrI distributions across the SP–PT scenarios. The boxplots highlight the much wider dispersion and larger shifts associated with low-PT networks, and show how these effects are further intensified when SP is also low, confirming the patterns observed in Table 2 for both positive and negative correlations.

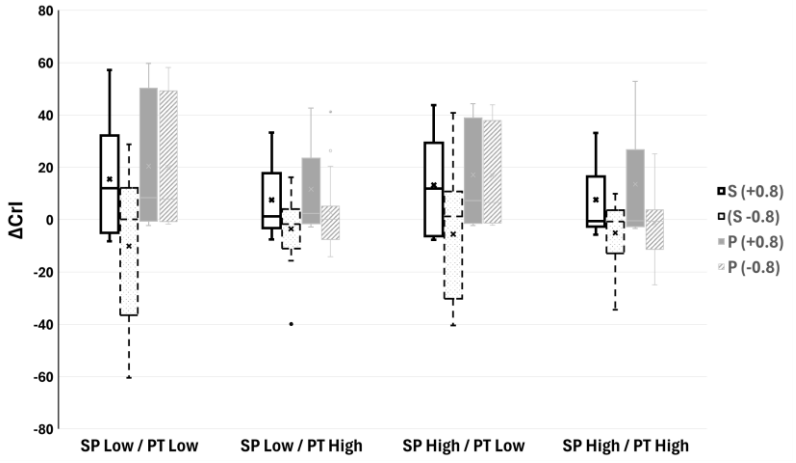


Figure 1. Distribution of ΔCrI across the four SP–PT scenarios, comparing positive (+0.8) and negative (–0.8) correlation.

3. Conclusions

In this paper, we have studied how correlations in project activity durations affect project uncertainty and activity risk-sensitivity indexes. In line with previous literature, we confirm that positive/negative correlations increase/decrease total project variance, with larger effects observed in projects with low SP and PT.

We take one step further by investigating how correlations and project network topology influence the values of risk-sensitive indexes, especially CrI. Both SP and PT help explain the observed differences in their variations, with low SP and PT values implying greater variability of the indexes.

According to our results, project managers should reflect during the project planning phase on potential sources of correlation between activity durations and consider, at least qualitatively, how their presence may affect risk control and monitoring practices during the execution phase.

4. Funding

This project has been partially funded by the Regional Government of Castile and Leon (Spain) under the Regional Accredited Research Groups Support Program, with grant VA042G2.

References

Acebes, F., De Antón, J., Villafáñez, F., & Poza, D. (2023). A Matlab-Based Educational Tool for Quantitative Risk Analysis. In *Lecture Notes on Data Engineering and Communications Technologies* (Vol. 160). https://doi.org/10.1007/978-3-031-27915-7_8

Acebes, F., & Pajares, J. (2025). *Project Tightness Index (PT): A Concise Metric of Schedule Tightness*. Working paper. https://uvadoc.uva.es/handle/10324/78733

Acebes, F., Pajares, J., Galán, J. M., & López-Paredes, A. (2014). Exploring the Influence of Seasonal Uncertainty in Project Risk Management. *Procedia - Social and Behavioral Sciences*, *119*, 329–338. https://doi.org/10.1016/J.SBSPRO.2014.03.038

Banerjee, A., & Paul, A. (2008). On path correlation and PERT bias. *European Journal of Operational Research*, *189*(3), 1208–1216. https://doi.org/10.1016/J.EJOR.2007.01.061

Demeulemeester, E., Vanhoucke, M., & Herroelen, W. (2003). RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling*, *6*(1). https://doi.org/10.1023/A:1022283403119

Edmondson, A. C., & Nembhard, I. M. (2009). Product development and learning in project teams: The challenges are the benefits. *Journal of Product Innovation Management*, *26*(2). https://doi.org/10.1111/j.1540-5885.2009.00341.x

Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*(2), 289–306. https://doi.org/10.1016/J.EJOR.2004.04.002

Hulett, D. T. (1996). Schedule Risk Analysis Simplified. *PM Network*, (610).

Kaut, M., Vaagen, H., & Wallace, S. W. (2021). The combined impact of stochastic and correlated activity durations and design uncertainty on project plans. *International Journal of Production Economics*, *233*. https://doi.org/10.1016/j.ijpe.2020.108015

Krishnan, V., Eppinger, S. D., & Whitney, D. E. (1997). A model-based framework to overlap product development activities. *Management Science*, *43*(4). https://doi.org/10.1287/mnsc.43.4.437

MacCrimmon, K. R., & Ryavec, C. A. (1964). An Analytical Study of the PERT Assumptions. *Operations Research*, *12*(1). https://doi.org/10.1287/opre.12.1.16

Ökmen, Ö., & Öztaş, A. (2008). Construction Project Network Evaluation with Correlated Schedule Risk Analysis Model. *Journal of Construction Engineering and Management*, *134*(1), 49–63. https://doi.org/10.1061/(ASCE)0733-9364(2008)134:1(49)

Van Dorp, J. R., & Duffey, M. R. (1999). Statistical dependence in risk analysis for project networks using Monte Carlo methods. *International Journal of Production Economics*, *58*(1), 17–29. https://doi.org/10.1016/S0925-5273(98)00081-4

Vanhoucke, M. (2010). Using activity sensitivity and network topology information to monitor project time performance. *Omega*, *38*(5), 359–370. https://doi.org/10.1016/J.OMEGA.2009.10.001

Vanhoucke, M. (2013). The PERT/CPM Technique. In *Project Management with Dynamic Scheduling*. https://doi.org/10.1007/978-3-642-40438-2_2

Yang, I. (2007). Risk Modeling of Dependence among Project Task Durations. *Computer-Aided Civil and Infrastructure Engineering*, *22*(6), 419–429. https://doi.org/10.1111/j.1467-8667.2007.00497.x

BALANCING HUMAN & AGENTIC PROJECT MANAGEMENT: AUTONOMY, ACCOUNTABILITY, AND TRANSPARENCY

Cohen Y[1], Sadeh A[2]

[1]Faculty of Industrial Engineering, Afeka College of Engineering, Israel
e-mail: yuvalc@afeka.ac.il

[2]Faculty of Industrial Engineering and Technology Management, Holon Institute of Technology, Israel
e-mail: sadeh@hit.ac.il

Keywords: AI Agents, Project Governance, Autonomy, Accountability, Transparency, Human–AI Collaboration.

1. Introduction

Project management increasingly incorporates AI agents capable of autonomous decision-making in resource allocation, risk assessment, and schedule optimization (Choudhury et. al. 2024). However, research reveals a critical gap: while AI's technical capabilities continue to advance, organizations struggle to govern agentic systems that simultaneously maximize efficiency, preserve accountability, and maintain trust (Müller et al., 2024; Floridi & Cowls, 2024). We address this gap through a theory-grounded conceptual model. Current PM literature treats autonomy, accountability, and transparency as independent variables (Endsley & Connors, 2023; Ritala et al., 2024). We propose that they are interdependent dimensions requiring dynamic balance. The key insight is that AAT Equilibrium, not the maximization of individual dimensions, predicts project outcomes.

This paper introduces the Agentic Cognitive Workflow (ACW) Model for optimizing human–AI collaboration in project management by balancing autonomy, accountability, and transparency (AAT). The model proposes that the AAT Equilibrium dynamic balance, calibrated to the project context, predicts outcomes better than maximizing individual dimensions. ACW specifies a closed-loop governance workflow that repeatedly: (1) profiles the decision context and project risk drivers (project complexity, decision uncertainty, task interdependence, and cognitive load); (2) sets graduated autonomy gates from “suggest-only” to “decide-and-implement” based on routineness and stakes; (3) assigns explicit decision rights, audit trails, and escalation paths to preserve accountability; and (4) provisions multi-layer transparency (role-appropriate explanations and confidence/uncertainty cues) to support justified decisions and calibrated reliance. Two mediating mechanisms operationalize the model in practice: Cognitive Affordance Alignment (CAA), which ensures users can accurately perceive and act on AAT features, and a Trust Symmetry Index (TSI), which aligns perceived agent capability with actual capability to prevent over- or under-reliance.

We propose that project outcomes are best predicted by the **AAT equilibrium** achieved through ACW’s adaptive cycle—not by maximizing any single dimension—and we present propositions and an empirical research agenda to test context-contingent AAT configurations, CAA/TSI effects, and adaptive AAT reconfiguration over time.

2. Theoretical Framework: The Autonomy Accountability Transparency Triad

We first identified the major dimensions that determine the types of work assigned to humans vs. agents in large projects: Autonomy, Accountability, and Transparency (AAT). We will describe as follows:

Autonomy (A): The degree to which AI agents can independently make decisions without human approval. Operationalised through graduated autonomy ranging from "suggest-only" to "decide-and-implement," dynamically adjusted based on task routineness, decision stakes, and cognitive load.

Accountability (Acc): Clarity about who is responsible for AI-supported decisions and what mechanisms exist to audit or contest them. The clarity and enforceability of responsibility for AI-assisted decisions. Implemented through human oversight structures, audit trails, decision rights assignment, and escalation procedures that maintain visibility of decision provenance.

Transparency (T): Extent to which users understand why the AI produced a recommendation, based on explanation quality and clarity. The interpretability of AI decision-making processes to

stakeholders. Achieved through multi-layered explainability tailored to user expertise, trust calibration mechanisms, and explicit confidence bound communication.

These dimensions are affected by various project risks and the organisational AI profile as described in Figure 1.

An AI profile naturally aligns with the team's AI literacy and culture. High literacy leads to higher trust in AI deployment and higher levels of autonomy. Lower literacy levels lead to lower independence. Organisational AI maturity is an essential factor that includes team literacy as a minor component, but it differs in its broader perspective.

Both factors are affecting the mediation of two novel constructs/measurements that we propose as follows:

Cognitive Affordance Alignment (CAA): The extent to which system design enables humans to accurately perceive and act upon A, Acc, T features. High CAA ensures design intent translates to user behaviour.

Trust Symmetry Index (TSI): Alignment between actual and perceived AI capabilities. TSI prevents "surprise failures" that destabilise trust despite continued transparency efforts.

The effects of Team AI literacy / prior experience on two constructs

High literacy is associated with higher baseline perception of affordances (CAA) and better trust calibration (higher TSI) for the same AAT and UI.

Low literacy requires more explicit training or guidelines, friendlier tools, and more active trust calibration cues (confidence indicators, performance feedback) for the same AAT. May reach acceptable CAA and TSI. We expect similar effects for high and low maturity. Figure 1 depicts the factors affecting the AAT policy.

2.1. Risk Factors: factors affecting the risks and AAT policy

In our research, “Risk Factors” are project- and environment-specific factors affecting AAT (such as risk level, regulatory demands, task complexity, and team AI literacy). These factors shape how a given autonomy, accountability, and transparency configuration, and its cognitive/trust

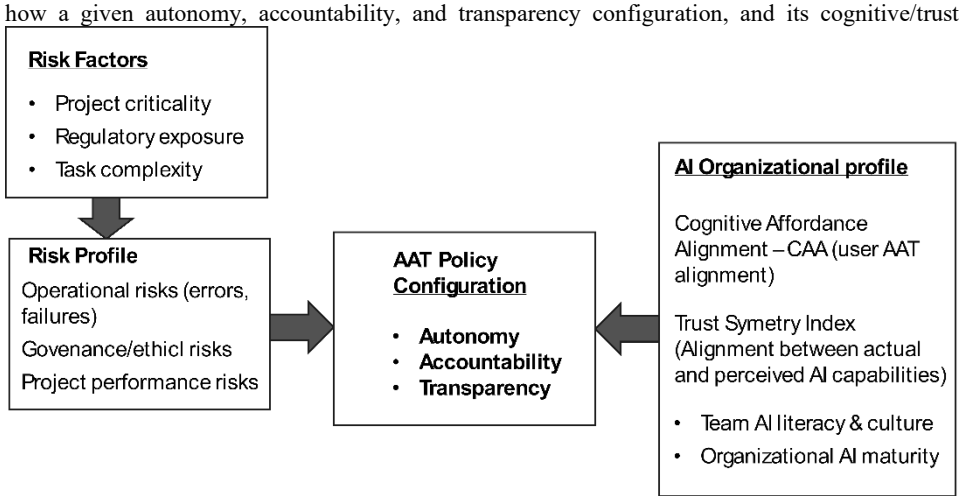


Figure 1. Factors affecting organizational Autonomy, Accountability & Transparency policy configuration

mechanisms influence human-AI collaboration outcomes. Risk Factors (e.g., project criticality, uncertainty, regulatory exposure, team AI literacy) shape both “where to set AAT” and “how high CAA/TSI needs to be.” The risk level is the main factor affecting the AAT policy, and our proposed new construction could reflect it:

- High-risk, high-regulation projects
  AAT: lower autonomy, stronger accountability, high transparency.
  CAA: interfaces strongly highlight manual approval, escalation routes, and justification logging.
  TSI: over-trust is especially dangerous; acceptable TSI thresholds are higher
- Low-risk, routine projects
  AAT: higher autonomy, lighter accountability, sufficient but not exhaustive transparency.

CAA: emphasis on making “auto mode” clearly visible and easily controllable.

TSI: Miscalibration is less catastrophic; you may tolerate some over-trust to gain efficiency.

We identify 4 different factors that determine the risk level and affect the A, A, T configuration.

1. Project Complexity (Pc): Technical, organizational, and environmental interdependence. Higher complexity requires lower autonomy due to non-linear risk interactions.
2. Decision Uncertainty (DU): Information scarcity and environmental volatility. Higher uncertainty demands dynamic uncertainty quantification in explanations.
3. Task Interdependence (TI): Functional linkage between project activities. Higher interdependence requires coordination mechanisms for autonomous AI decisions.
4. Cognitive Load (CL): Mental effort required of managers. Higher load reduces the effectiveness of complex explanations.

3. Proposed Measurement Scales

using 5-point Likert scales (1 = strongly disagree, 5 = strongly agree). Each construct includes example items plus a brief illustration in a PM setting.

AI autonomy (task-level decision authority):

Example items to rank using a Likert scale (1-5). (per specific task, e.g., scheduling, risk analysis):

“The AI agent can make changes to the project schedule without my prior approval.”

“The AI agent independently adjusts task priorities when new information arrives.”

“The AI agent initiates risk alerts or mitigation actions on its own.”

“The AI agent learns from past project data and changes its recommendations without being explicitly told to do so.”

Short example: In a software project, the scheduling agent automatically reassigns developers when it detects delays, without waiting for the project manager to confirm each change.

Accountability (clarity of responsibility and recourse)

Example items: to rank using a Likert scale (1-5).

“It is clear who is ultimately responsible if an AI-supported project decision causes a problem.”

“There are documented procedures for reviewing and approving important decisions suggested by the AI agent.”

“If I disagree with an AI recommendation, I know how to challenge or override it.”

“The organization keeps records that allow us to trace which decisions were influenced by the AI agent.”

Short example: A risk-scoring agent flags suppliers; the PMO guideline specifies that the category manager, not the AI, is accountable for the final supplier selection and must sign off on exceptions.

Transparency/explainability (quality and clarity of system explanations)

Example items (inspired by SCS and explanation “goodness” work):

“When the AI agent recommends an action, I can understand the main reasons behind it.”

“The AI’s explanation focuses on the most relevant project factors (e.g., cost, time, risk).”

“I can usually tell how changes in input data (e.g., effort estimates) would affect the AI’s recommendation.”

“The explanations from the AI agent are clear enough for me to justify decisions to stakeholders.”

Short example: The AI suggests delaying a milestone and shows that two critical tasks are overloaded and a key resource will be unavailable; the PM can easily explain this to the steering committee.

.

4. Propositions for further research

We propose the following 7 propositions(P1-P7) to be examined in subsequent research.

- P1 - Autonomy Effects: Graduated autonomy positively influences decision quality when aligned with task routineness but exhibits a non-monotonic relationship (P1.1). Adaptive authority allocation improves project agility in complex environments (P1.2).

- P2 - Accountability Effects: Clear oversight structures increase trust (P2.1). Audit trails enable high-stakes delegation (P2.2). Clarity about decision rights improves efficiency when aligned with stakeholder expectations (P2.3).
- P3 - Transparency Effects: Multi-layered explainability improves calibrated trust (P3.1). Explicit confidence bounds reduce over-reliance (P3.2). Interactive explanations outperform static ones in high-stakes decisions (P3.3).
- P4 - Contextual Moderation: Project complexity, decision uncertainty, task interdependence, and cognitive load each moderate optimal A, Acc, T configurations (P4.1–P4.4).
- P5 - AAT Equilibrium (Synergistic): Joint AAT balance generates super-additive outcomes (P5.1). AAT equilibrium mediates context-outcome relationships (P5.2). Adaptive AAT reconfiguration enables sustained superior performance (P5.3).
- P6 - Integration Mechanisms. CAA mediates the relationship between design features and trust (P6.1–P6.2).
- P7 - TSI mediates the relationship between transparency and sustained trust (P7.1–P7.2).

Conclusion

The proposed Agentic Cognitive Workflow model advances project management theory by positioning autonomy, accountability, and transparency as interdependent dimensions that must be jointly configured rather than optimized in isolation when deploying AI agents in projects. By articulating six propositions that connect different AAT configurations to project performance, risk, and ethical outcomes, the paper provides a structured basis for moving beyond descriptive accounts of AI in PM toward explanatory and predictive theory. A key implication is that "more autonomy" is not universally beneficial; instead, effective configurations depend on project uncertainty, criticality, and the maturity of both human and technical systems, calling for dynamic governance that can adjust delegation and oversight over time.

For practitioners and PMOs, the ACW model offers a conceptual blueprint for designing agentic project environments in which AI systems act as accountable collaborators embedded within clear governance boundaries. Concretely, the model suggests specifying levels of agent autonomy by task type, defining explicit accountability chains for AI-originated actions, and implementing transparency mechanisms such as explainable recommendations and traceable decision logs—that enable human supervisors to calibrate trust and intervene when needed. Future research should operationalize the ACW constructs, develop measurement instruments for AAT equilibrium, and empirically test the propositions through case studies, surveys, and experiments across diverse project contexts, ultimately informing standards and guidelines for responsible agentic AI in project management.

References

Choudhury, P., et al., 2024, "When AI becomes an agent of the firm". Journal of Management Studies, https://doi.org/10.1111/joms.13274

Endsley, M. R., & Connors, E. S. (2023). Supporting human-AI teams: Transparency, explainability, and situation awareness. Computers in Human Behavior, Vol. 140, 107574.

Floridi, L., & Cowls, J. (2024). "Transparency and accountability in AI systems", Frontiers in Human Dynamics, Vol. 6, 1421273.

Müller, R., Locatelli, G., Holzmann, V., Nilsson, M., & Sagay, T. (2024). "Artificial intelligence and project management: Empirical overview, state of the art, and guidelines for future research", Project Management Journal, Vol. 55, No. 1, pp. 3–15.

Ritala, P., et al., 2024, "Developing industrial AI capabilities: An organisational learning perspective", Technovation, Vol. 130, 102923.

Extended application of Kalman Filter Forecasting Method (KFFM) under increasing uncertainty

Yaodong Wang[1], Izel Unsal Altuncan[1] and Mario Vanhoucke[1,2,3]

[1] Ghent University, Belgium
yaodong.wang@ugent.be
izel.unsalaltuncan@ugent.be
[2] Vlerick Business School, Belgium
[3] University College London, UK
mario.vanhoucke@ugent.be

Keywords: Project forecasting, Project simulation, Kalman Filter.

1 Introduction

Accurately estimating project durations is the cornerstone of effective project management. Traditional estimation methods such as Earned Value Management (EVM) can provide accurate results when sufficient progress data are available but tend to be less reliable in the early phases of a project when limited actual information exists. To address this challenge, [Kim and Reinschmidt, 2010] introduced the Kalman Filter Forecasting Method (KFFM), which they argue can be applied from the very beginning of a project without significant loss of forecasting accuracy. However, three important gaps in that existing research on the KFFM limit the generalizability of these results. First, the proposed KFFM methodology is evaluated only on two case studies, which is not sufficient to assess the applicability across projects with different characteristics. Second, the existing research does not provide the accuracy results over the complete project horizon, which prevents capturing the performance of the KFFM for varying levels of percentage completion of the projects. Third, the level of uncertainty in the test projects used in the existing research lacks variability which limits insights into how the higher uncertainty affect the accuracy. In this study, we address these gaps by applying KFFM to a large and diverse set of project instances and analyzing the results for varying levels of percentage completion under increasing duration uncertainty.

2 Kalman Filter forecasting method

KFFM is a dynamic project forecasting method applied at discrete tracking periods $k = 0, 1, 2, ..., K$, where K represents the total number of tracking periods. It is based on a recursive algorithm that continuously refines predictions of the final project duration by updating the initial estimates for the project state with the progress observed at each tracking point.

The method relies on three groups of variables. The first group consists of the *state variables*, which characterize the progression of the project. The second group consists of the *covariance variables*, which quantify the uncertainty associated with the state variables. The third group includes the *system matrices*, namely the state-transition function and the observation matrix. These variables are used across the five steps of the KFFM procedure, which are summarised below and illustrated in Figure 1.

Step 1 (Initialization): The procedure starts at $k = 0$ before the project begins, and specifies the initial state variables, their initial covariance, and the noise parameters (process noise (Q) and measurement noise (R)). These values define the *prior* belief about the project's progress before any predictions or observations are incorporated into the model.

Step 2 (Predict): In this step, the tracking period is increased to $k = k + 1$ and initial state and covariance variables from Step 1 are propagated through the next tracking period using the state transition function, yielding the a *prior* estimate of progress. These values are referred to as the predicted state and predicted covariance and represent the model-based prediction *prior* to observing project progress.

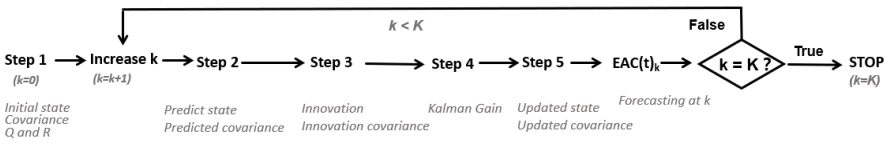


Fig. 1: The KFFM forecasting process

Step 3 (Innovation): In this step, the predicted state variable from Step 2 is transformed into a predicted progress variable using the observation matrix. Then, a new variable, innovation, is computed as the difference between the observed project progress and this predicted progress variable. In parallel, the innovation covariance is obtained by propagating the predicted covariance from Step 2 through the observation matrix and adding the measurement noise R.

Step 4 (Kalman Gain): In this step, the Kalman Gain vector is computed by combining the predicted covariance from Step 2 with the innovation covariance from Step 3. The Kalman Gain determines how much weight is given to the new observation from Step 3 relative to the model-based prediction from Step 2. A higher gain places more weight on the observed progress from Step 3, whereas a lower gain places more weight on the predicted state from Step 2.

Step 5 (Update): Finally, the covariance is updated using the Kalman Gain, the predicted covariance from Step 2, and the observation matrix. The predicted state from Step 2 is then corrected with the Kalman Gain and the innovation to obtain the state estimate for tracking period k. This state estimate is subsequently converted into a forecast of the final project duration at tracking period k, denoted as $EAC(t)_k$.

After each forecast, the KFFM procedure controls whether all tracking periods are processed. If additional tracking periods remain, it returns to Step 2 and repeats the update for the next period. Once after the forecast for the final tracking period K has been obtained, the KFFM procedure terminates.

3 Methodology

In this study, the KFFM is applied to a large artificial project dataset consisting of 900 project networks derived from the RanGen2 project network generator proposed by [Vanhoucke et al., 2008]. The project networks are generated by varying the network topologies based on the Serial/Parallel (SP) indicator. For each SP value in the set $\{0.1, 0.2, \ldots, 0.9\}$, 100 project instances are generated.

Subsequently, these project instances are subjected to two types of Monte-Carlo simulations namely, static and dynamic simulations.

Static simulations are applied at Step 1 before the project starts to generate values for the process noise covariance (Q). The mean and variance of the simulated project durations are iteratively processed using the KFFM procedure until convergence, yielding the converged estimate of Q for each instance. Once the procedure is finished, the resulting Q is recorded as the final value for that project.

Dynamic Monte-Carlo simulations are performed to imitate real project execution under varying levels of duration uncertainty. During these dynamic simulations, the uncertainty of each activity duration is represented using a triangular distribution with three scenarios representing low, medium and high duration uncertainty levels. For each project, periodic progress data is measured at nine tracking periods ($K = 9$) using earned value management metrics and their extension to Earned Schedule (ES) [Lipke, 2003]. At each tracking period, a prediction for the final project duration is generated by applying the KFFM steps from Step 2 to Step 5 iteratively.

4 Computational experiments and preliminary results

During the computational experiments, we apply the KFFM procedure to 900 artificial project instances generated under varying levels of activity duration uncertainty. In the first experiment, we investigate how KFFM accuracy is affected by the network topology, measured by SP. In the second experiment, we examine the impact of the project stage, expressed as the percentage completion at the time of forecasting. For both experiments, the analyses are conducted across different levels of duration uncertainty. The accuracy of KFFM is assessed using the Mean Absolute Percentage Error (MAPE) of the forecasts, benchmarked against the ES method.

Dynamic simulation	Serial/Parallel Indicator (SP)		
	0.2	0.5	0.8
Low uncertainty	3.8	4.2	3.5
Medium uncertainty	8.9	10.3	10.3
High uncertainty	12.7	14.8	15.4

Table 1: Impact of SP on the accuracy

Table 1 presents the results for the first experiment and indicate that the impact of network topology is most pronounced at higher levels of duration uncertainty. For serial projects (higher SP values) under medium and high uncertainty, the accuracy of the method decreases, as shown with higher MAPE values. For low uncertainty projects, the impact of SP on forecasting accuracy is observed only marginally.

For the second experiment, Figure 2 presents the forecasting accuracy of KFFM and the ES method, across different project completion levels, on average over three duration uncertainty settings. The results show that, at early stages, KFFM outperforms ES in terms of forecasting accuracy, consistent with findings reported in previous KFFM studies. However, our results further indicate that the performance of the averaged KFFM gradually decreases over time. After around 55% project completion (PC), the averaged ES outperforms KFFM,

indicating that KFFM becomes less reliable in the later phases regardless of the uncertainty level. These findings suggest that higher overall accuracy may be achieved by using KFFM during approximately the first half of the project and then switching to ES thereafter.

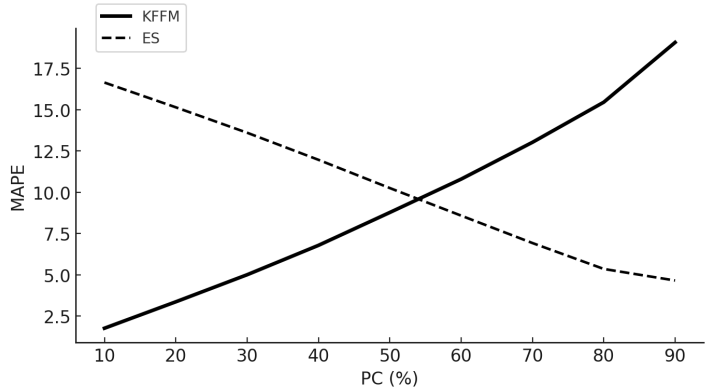


Fig. 2: KFFM vs. ES MAPE across PC levels

These preliminary findings indicate that the accuracy of the KFFM depends on the level of uncertainty in the project execution and the project completion percentage at the time of forecasting. Furthermore, the results show that the impact of the network topology is mostly pronounced under higher uncertainty levels observed during the project execution.

To investigate this relationship more rigorously, the future analysis will test the models under multiple uncertainty settings, examining how the relative performance of ES and KFFM shifts as uncertainty varies. In addition to the artificial experiments, the evaluation will be further extended to empirical project data to validate whether the observed patterns hold in practical applications. Furthermore, to address the limitations of KFFM, an extended version, Extended Kalman Filter Forecasting Model (EKFFM), is developed. The forecasting performance of ES, KFFM, and EKFFM is then compared under various uncertainty conditions.

References

Kim and Reinschmidt, 2010. Kim, B.-C. and Reinschmidt, K. F. (2010). Probabilistic forecasting of project duration using kalman filter and the earned value method. *Journal of Construction Engineering and Management*, 136(8):834–843.

Lipke, 2003. Lipke, W. (2003). Schedule is different. *The Measurable News*, pages 31–34.

Vanhoucke et al., 2008. Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., and Tavares, L. (2008). An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, 187:511–524.

15-15 Apr 2026

G - Complexity and approximation

Complexity analysis for the EVCSP with active charger constraints

Arthur Mazeyrat[1], Ammar Oulamara[2], Tifenn Rault[1] and Ameur Soukhal[1]

[1] University of Tours, LIFAT EA 6300, 64 avenue Jean Portalis, 37200 Tours, France
athur.mazeyrat@etu.univ-tours.fr
tifenn.rault,ameur.soukhal@univ-tours.fr
[2] University of Lorraine, LORIA - UMR 7503, Campus Scientifique - BP 239, 54506 Vandoeuvre-les-Nancy, France
ammar.oulamara@loria.fr

Keywords: Scheduling, Electric Vehicles, Complexity, Heuristic.

1 Introduction

The Electric Vehicle Charging Scheduling Problem (EVCSP) can be stated as follow: we consider a charging station with m identical chargers, each delivering a power w, and a global grid capacity w_G that limits the total power that can be supplied simultaneously by all chargers. Each electric vehicle (EV) charging request is represented by an interval $[r_i, d_i]$, where r_i is the EV arrival time and d_i its departure time, together with a required energy e_i expressed as charging duration $p_i = \frac{e_i}{w} \leq d_i - r_i$. Any charged vehicle must be plugged to a single charger during its whole interval $[r_i, d_i]$. The charging may be preemptive. The objective is to maximize the number of accepted charging requests. This problem is known to be NP-hard (Zaidi et al. 2022) when several chargers may be activated simultaneously. However, the complexity of the special case in which at most one charger can be activated at any time remains open (Zaidi et al. 2022), i.e. $w \leq w_G < 2w$. We can view this version of the problem as adding a capacity of m parking spots each having a charger, while at most one charger can be activated at any time.

When at most one charger can be activated at each time, we study both cases: the case where the number m of chargers is a part of the instance and the case with fixed m. The charging may be preemptive and we also suppose that any charged vehicle occupies a charger during its whole interval.

We prove that this problem is NP-hard via a reduction from the Equal-Cardinality Partition Problem (ECPP) and we develop an heuristic achieving excellent results.

Using Grahams three-field notation (Graham et al. 1979), the problem $1 \mid pmtn, r_j \mid \sum U_j$ is very close to ours, except that it does not incorporate active charger constraints. For this case, polynomial-time algorithms are known: Lawler's algorithm of complexity $O(n^5)$ (Lawler 1990), Baptiste's improvement to $O(n^4)$ (Baptiste 1999), and Vakhania's $O(n^3 \log n)$ algorithm (Vakhania 2009).

2 NP-hardness of the EVCSP with active charger constraints

We show that the problem EVCSP when the number m of chargers is a part of the instance and at most one charger can be activated at each time is NP-hard. The proof is given by reduction from the Equal-Cardinality Partition Problem.

Equal-Cardinality Partition Problem (ECPP):

Given a set I of $2n$ distinct integers a_i, $i = 1, \ldots, 2n$, and an integer B where $\sum_i a_i = 2B$. The ECPP consists in finding a partition of I into two subsets I_1 and I_2, such that: $I = I_1 \cup I_2$, $|I_1| = |I_2| = n$ and $\sum_{i \in I_1} a_i = B$. This problem is NP-hard (Garey & Johnson 1979).

Reduction to EVCSP:

We now reduce any instance of ECPP to an instance of EVCSP. Given an instance I of ECPP, we build an instance I' of EVCSP consisting in $4n$ charging requests defined in Table 1. A visual representation of a typical instance for $2n = 10$ is shown in Figure 1. We fix the number of chargers to $m = n$. The instance I' has an unique optimal solution of value $2n$, such that finding such solution automatically yields an Equal-Cardinality Partition of I.

Table 1. Instance of EVCSP

$\forall i \in [\![1, 2n]\!]$	L-jobs	R-jobs
r_i	0	$(n+1)B + n - i$
d_i	$(n+1)B + n - i$	$(n+1)B + (n+1)^2 B$
p_i	$(n+1)a_i$	$\frac{2(n+1)B}{n} - (n+1)a_i + (n+1)B$

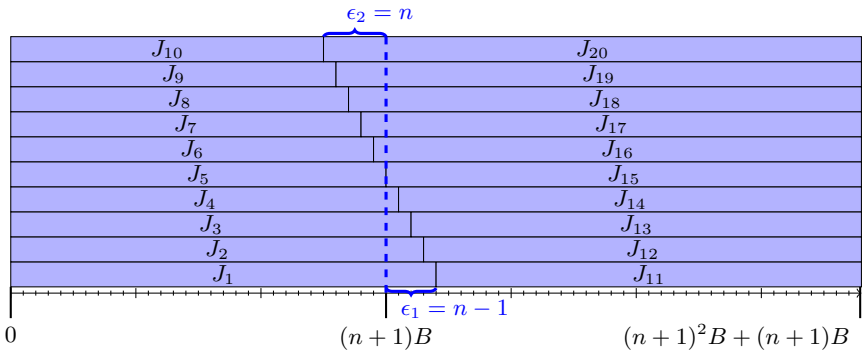


Fig. 1. Instance of EVCSP for $2n = 10$

Key Properties

- Any feasible EVCSP solution selecting $2n$ requests must pick exactly n requests in the L-jobs part and n requests in the R-jobs part due to the number of chargers.
- Any Equal-Cardinality Partition I_1, I_2 yields an optimal solution of our problem. In other words, if $a_i \in I_1$, then requests i and $(i + 2n)$ in I' are selected for charging.
- The sum of the selected L-jobs charging durations requests cannot exceed $(n+1)B$, as all L-jobs charging durations are multiple of $(n + 1)$ and that $\epsilon_1 \leq n - 1$.
- For similar reasons, the sum of charging durations of the selected R-jobs cannot exceed $(n + 1)^2 B$.
- There exists a perfect matching between L-jobs and R-jobs in any optimal solution, such that the matching is decreasing on the right side: each selected L-job i is matched to a selected R-job $(j + 2n)$ with $j \leq i$, $j \in [\![1, 2n]\!]$.
- Thanks to such a decreasing perfect matching, we can prove that the sum of charging durations of selected L-jobs must be bigger than $(n + 1)B$.
- Thus, any optimal solution of EVCSP corresponds exactly to a valid Equal-Cardinality Partition of I, and vice versa.

Theorem 1. *The EVCSP with m chargers, where m is part of the instance, and at most one charger can be activated at each time is NP-hard in the ordinary sense.*

3 EVCSP under a fixed number of identical chargers

In real life applications, the number of chargers is limited to a certain fixed m. Moreover, it may happen that some chargers are unavailable during a certain period of time. We can consider this version of the problem as having variable charger availability constraints over time horizon. If we define a function f that gives for any time t the number of available chargers, we then have that $f(t) \leq m,\ \forall t$.

We showed that this problem is polynomial in n for any fixed m by proving that the enumeration of all solutions is polynomial.

4 Heuristic approach

We propose to first select subsets of vehicles that can be assigned to chargers (at most m at each time). Since the single-charger EVCSP with an unlimited number of chargers can be solved in polynomial time, once such a subset is selected, the instance can be solved using any polynomial-time algorithm from the literature (Lawler 1990), (Baptiste 1999), (Vakhania 2009).

In order to select such a subset, we associate a score with each vehicle. By denoting $\ell_j = d_j - r_j$ the length of request j, we define the score function S as:

$$S(j) = -\ell_j p_j$$

This score favors small jobs in terms of length and energy request. We now construct a solution I' by iteratively selecting the highest-score request J_{i^*} and adding it to I' if it can be scheduled with the current I', i.e. if the number of intervals in $[r_{i^*}, d_{i^*}]$ does not exceed the maximum capacity m; otherwise, it is discarded.

5 Computational experiments

We generated random instances as follows. For any number n of vehicles, we first generated n release dates r_j uniformly in $[0, T-1]$, n due dates d_j uniformly in $[r_j+1, T]$, and then generated a charging duration p_j uniformly in $[1, d_j - r_j]$, with $T = 10000$. The number of chargers was set to $m = \frac{n}{5}$. For each $n \in \{50, 100, 200, 300\}$, 100 instances were generated. In addition, we generated 100 hard instances of size $n = 100$ by generating an instance of ECPP and then translating it into an instance of EVCSP. Optimal solutions were obtained via an efficient MILP formulation (Zaidi et al. 2022).

For a given instance I, let f_I^* denote the optimal objective value and f_I^A the objective value obtained by the heuristic, and the absolute gap $g_I^A = f_I^* - f_I^A$ (axis x in Figure 2).

On the random instances (Figure 2), the heuristic consistently achieves near-optimal solutions, with performance improving as n increases. In Table 2, we report the relative gaps.

Table 2. Relative gap of random instances

n	50	100	200	300
Relative gap (%)	1.8	0.2	0.06	0.0

On the hard instances with $n = 100$, while the optimal objective value is $f_I^* = 2n$, the heuristic consistently returns $f_I^A = 2n-1$, providing strong performance even in challenging cases.

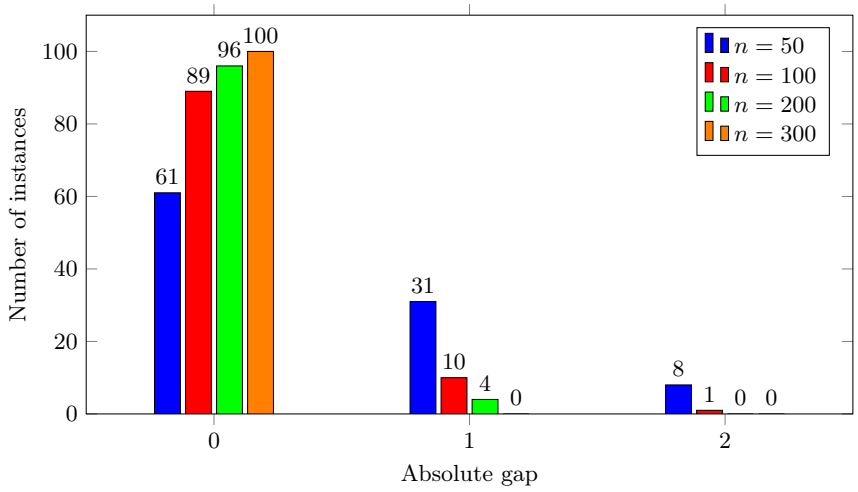


Fig. 2. Gap of 100 random instances generated for $n \in \{50, 100, 200, 300\}$

6 Conclusion and Perspectives

We have proved that the EVCSP with one active charger constraints and m chargers is NP-hard when m is part of the instance.

We developed an efficient heuristic that achieves excellent results while remaining polynomial, with complexity $O(n^3 \log n)$ using Vakhania's algorithm (Vakhania 2009).

It would be interesting to adapt the heuristic to handle the *locality* of any request. It would also be interesting to investigate whether the proposed heuristic can be formally characterized as an approximation algorithm for the studied problem.

Acknowledgements

We thank the Région Centre-Val de Loire for its support.

References

Baptiste, P. (1999), "An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs", *Discrete Applied Mathematics*, 24, pp. 175–180.

Garey, M.R. and Johnson, D.S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco.

Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1979), "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey", *In*: Hammer, P.L., Johnson, E.L. and Korte, B.H. (eds.), *Discrete Optimization II*, Annals of Discrete Mathematics, vol. 5, Elsevier, pp. 287–326.

Lawler, E.L. (1990), "A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs", *Annals of Operations Research*, 26, pp. 125–133.

Vakhania, N. (2009), "Scheduling jobs with release times preemptively on a single machine to minimize the number of late jobs", *Operations Research Letters*, 37, pp. 405–410.

Zaidi, I., Oulamara, A., Idoumghar, L. and Basset, M. (2022), "Maximizing the number of satisfied charging demands of electric vehicles on identical chargers", *Omega*, 127, pp. 103–106.

Single machine scheduling of low-frequency radio-astronomical observations

Maher Mallem[1] and Frédéric Vivien[1]

[1]Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342, Lyon cedex 07, France
maher.mallem@ens-lyon.fr, frederic.vivien@inria.fr

Keywords: complexity, scheduling, single machine, release date, deadline, astronomy.

1 Introduction

This work tackles the scheduling of radio-astronomical observations, in the context of the Square-Kilometre Array Observatory project (SKAO) and in partnership with the ECLAT laboratory. NenuFAR is a cutting-edge low-frequency radio telescope at the Nançay Radioastronomy Observatory in France. Unlike optical telescopes, radio telescopes can be used in the daytime as well as at night. Still, the altitude of observed celestial bodies should ideally be high enough for responses to be quasi-isotropic — e.g., at least 20° with NenuFAR (Zarka *et. al.* 2020). Because the Earth revolves around itself, plotting the altitude of a celestial body over time yields a near-sinusoidal signal on a twenty-four hour period. Therefore, setting an altitude lower bound leads to a daily availability window for each celestial body, with no specific preferred observation time within this range. Formally, given a day length $L \in \mathbb{R}_{>0}$, such observations can be modeled as a set of n jobs $\mathcal{J} = \{1, \ldots, n\}$ with, for each job j in $\mathcal{J}$, a processing time $p_j \in \mathbb{R}_{>0}$, a daily release date $r_j \in [0, L)$ and a daily deadline $d_j \in [0, L)$.

Now, low-frequency signals of such distant objects are typically faint, and thus require long observations which cannot be easily preempted. In particular, although NenuFAR is made of close to two thousand antennas, a single celestial body is usually observed at a time, with all the antennas pointing to it in order to amplify the retrieved signal. So, this setting can be modeled as scheduling non-preemptive jobs on a single machine. Plus, since this machinery is expensive even when not in use and burdensome to shut down/set up, the primary goal is to minimize the idle time which, in our setting, is equivalent to minimizing the makespan. We denote the resulting scheduling problem by $1|periodic_L(r_j, d_j)|C_{\max}$. Surprisingly, to the best of our knowledge, this does not seem to correspond to any previously studied scheduling problem. As such, in this contribution, we propose to investigate the (parameterized) complexity of this scheduling problem, and unveil connections to more established scheduling settings.

Remark 1. W.l.o.g. one can assume that for all jobs j, $p_j \in (0, L]$. Indeed, if $p_j > L$, then the extra multiples of L in p_j correspond to full days of processing job j no matter the starting time of job j. So these extra multiples can be removed from p_j before solving the problem, then they can be inserted back with no issue.

2 Contribution

2.1 Computational complexity

First, we consider the computation of a schedule of minimum makespan OPT. According to the single machine setting with regular time windows $1|r_j, d_j|C_{max}$, our problem is (strongly) NP-complete even on a single day (Lenstra *et. al.* 1977). It even generalizes machine minimization problem $P|r_j, d_j| \min(m)$, where each machine is interpreted as a day. In fact, in our setting, makespan minimization is equivalent to day minimization.

Theorem 1. *If $D \in \mathbb{R}^+$, then decision problem $1|periodic_L(r_j, d_j)|C_{\max} \leq D$ can be reduced to decision problem $1|periodic_L(r_j, d_j)|C_{\max} \leq dL$, where $d = \lceil D/L \rceil$.*

Proof. (Sketch.) If D is a multiple of L, then nothing needs to be done. Otherwise, let $\rho = D \bmod L$. We add a fill job f of processing time ρ, daily release date $(L - \rho)$ and daily deadline 0. Given a schedule τ of makespan at most dL featuring this extra job f, the sequence of scheduled jobs in τ can be written as AfB. We propose a schedule with job sequence BA for the original schedule. Indeed, job f is necessarily completed at the very end of a day. So the parts Af and B of schedule τ can safely be permuted to obtain a schedule with job sequence BAf and makespan at most dL. Then job f can be removed to yield a schedule of makespan at most L for the original instance. □

In response to the NP-completeness of our problem, we propose the following single-exponential time algorithm.

Theorem 2. *$1|periodic_L(r_j, d_j)|C_{\max}$ can be solved in $\mathcal{O}(n \cdot 2^n)$ time.*

Proof. (Sketch.) We propose a dynamic programming algorithm computing the minimum makespan $M(S)$ over all job subsets $S \subseteq \mathcal{J}$. Given a job j and a time t, let $s_j[t]$ be the earliest available start time of job j which is greater than or equal to time t. We set $M(\emptyset) = 0$ and, if $S = \{j_1, \ldots, j_k\}$, then:

$$M(S) = \min_{1 \leq \ell \leq k} (s_{j_\ell}[M(S \backslash \{j_\ell\})] + p_{j_\ell}). \tag{1}$$

The 2^n subsets are considered by nondecreasing order of their size. Each subset takes $\mathcal{O}(n)$ time, with each value $s_{j_\ell}[M(S\backslash\{j_\ell\})]$ taking $\mathcal{O}(1)$ operations — provided that minimum makespan value $M(S\backslash\{j_\ell\})$ has been computed. □

We also show that the preemptive variant of our problem is polynomial-time solvable.

Theorem 3. *Preemptive $1|periodic_L(r_j, d_j)|C_{\max}$ is polynomial-time solvable.*

Proof. (Sketch.) Given a makespan threshold $D \in \mathbb{N}$, by Theorem 1 we consider an equivalent instance of decision problem $1|periodic_L(r_j, d_j)|C_{\max} \leq dL$ with $d = \lceil D/L \rceil$ and at most one additional job. We provide a translation of this instance into a maximum flow instance with $\mathcal{O}(n)$ nodes and $\mathcal{O}(n^2)$ edges. First, in $\mathcal{O}(n \log(n))$ time, we compute $u_1, \ldots, u_K$ the increasing sequence of release date and deadline values combined with values 0 and L. Then, on top of source s and sink t, we have two layers of nodes. For every job j in $\mathcal{J}$ we have a node a_j with an edge from source s of capacity p_j. And, for every k in $\{1, \ldots, K-1\}$, we have a node b_k with an edge of capacity p_j from every job j, the daily time window of which includes interval $[u_k, u_{k+1})$, and an edge to sink t of capacity $d \cdot (u_{k+1} - u_k)$.

Given a solution of this maximum flow instance with $\mathcal{O}(n)$ nodes and $\mathcal{O}(n^2)$ edges, a schedule can be obtained by greedily filling every daily time interval $[u_k, u_{k+1})$ one day at a time. Finally, a schedule of optimal makespan can be found by binary search on the value of D with initial lower bound $(\sum_{1 \leq j \leq n} p_j)$ and initial upper bound $(nL + \sum_{1 \leq j \leq n} p_j)$ (since there always is an available start time before waiting for a full day). □

Finally, note that our setting is reminiscent of the discrete Interval Scheduling problem, where a collection of arbitrary start times (i.e., not periodic and not necessarily consecutive) is given to each job. While scheduling equal-length jobs with three arbitrary start times per job is (strongly) NP-complete (Keil 1992), our setting can be solved in $\mathcal{O}(n \cdot \log(n)^2)$ time when time windows are tight — i.e., when their length is equal to the job processing time (Dereniowski and Kubiak 2010). This highlights the periodicity of job availabilities as a key property in our setting.

2.2 Approximation

Regarding approximation, the NP-completeness of our problem over a single day implies that there is no better than a 2-approximation algorithm — and thus no PTAS.

Theorem 4. *Unless* $\mathsf{P} = \mathsf{NP}$*, for every* $0 < \varepsilon < 1$ *there is no* $(2-\varepsilon)$*-approximation algorithm for problem* $1|periodic_L(r_j, d_j)|C_{\max}$.

Proof. (Sketch.) Let $\varepsilon > 0$. We reduce from NP-complete problem $1|r_j, d_j|C_{\max} \leq D$ and create a $(2-\varepsilon)$-gap between YES and NO instances. Given an instance $\mathcal{I}$ of the former problem, we set $L = (D+1) \cdot \lceil 1/\varepsilon \rceil$ and create an instance $\mathcal{I}'$ of $1|periodic_L(r_j, d_j)|C_{\max}$ featuring $\lceil 1/\varepsilon \rceil$ disjoint copies of $\mathcal{I}$. For i in $\{1, \ldots, \lceil 1/\varepsilon \rceil\}$, the i^{th} copy fits within daily time interval $[(D+1)(i-1), (D+1)i)$. Now, if $\mathcal{I}$ is a YES instance, then we replicate the solution in every copy to obtain a schedule with makespan at most L. Otherwise, at least one job from the latest copy must be completed after time $2L - (D+1)$. This induces a makespan gap between YES and NO instances which is lower bounded by $(2-\varepsilon)$. □

Now, whenever $OPT \geq L$, we propose a general way to adapt approximation algorithms for the machine minimization setting, at the cost of an extra $2(1 + L/OPT)$ factor.

Theorem 5. *Let* $\mathcal{A}'$ *be a* $f(n)$*-approximation algorithm for problem* $P|r_j, d_j|\min(m)$ *for some function* f*. Then one can build a* $[2(1 + L/OPT) \cdot f(n)]$*-approximation algorithm* $\mathcal{A}$ *for problem* $1|periodic_L(r_j, d_j)|C_{\max}$ *running in* $\mathcal{O}(\max(n, time(\mathcal{A}')))$ *time.*

Proof. Let $\mathcal{I} = \langle \mathcal{J}, L, p_j, r_j, d_j \rangle$ be an instance of $1|periodic_L(r_j, d_j)|C_{\max}$. By Remark 1, assume that all jobs j have processing time at most L. Consider instance $\mathcal{I}' = \langle \mathcal{J}, p_j, r_j, d'_j \rangle$ of $P|r_j, d_j|\min(m)$ where $d'_j = d_j + L$ if the time window of job j overlaps with time zero, and $d'_j = d_j$ otherwise. We propose algorithm $\mathcal{A}$ which computes instance $\mathcal{I}'$ from $\mathcal{I}$, computes the schedule τ' returned by algorithm $\mathcal{A}'$ on $\mathcal{I}'$, then proposes the following schedule τ: if τ' is a schedule on m' machines and job j is scheduled on machine $k \in \{0, \ldots, m'-1\}$ in τ', then $\tau(j) = 2kL + \tau'(j)$. Clearly, algorithm $\mathcal{A}$ runs in $\mathcal{O}(\max(n, time(\mathcal{A}')))$ time.

We show that algorithm $\mathcal{A}$ is a $[2(1+L/OPT) \cdot f(n)]$-approximation algorithm for problem $1|periodic_L(r_j, d_j)|C_{\max}$. Let τ_{OPT} be a schedule for instance $\mathcal{I}$ with makespan OPT. Then we can deduce a schedule τ'_{OPT} for instance $\mathcal{I}'$ on $\lceil OPT/L \rceil$ machines: if job j starts during day $i \in \{0, \lceil OPT/L \rceil - 1\}$, then we set: $\tau'_{OPT}(j) = \tau_{OPT}(j) - iL$. This means that value $\lceil OPT/L \rceil$ is lower bounded by OPT' the optimal number of machines for instance $\mathcal{I}'$.

Now, because schedule τ' uses at most $f(n) \cdot OPT'$ machines, the makespan of schedule τ is at most equal to $2L \cdot f(n) \cdot OPT'$. By the previous paragraph, this value is upper bounded by $2L \cdot f(n) \cdot \lceil OPT/L \rceil$, and thus by $2(OPT + L) \cdot f(n)$. □

Then, a $\mathcal{O}(\log(n))$-approximation algorithm for $1|periodic_L(r_j, d_j)|C_{\max}$ can be inferred from Theorem 5 by considering the $\mathcal{O}(\log(n))$-approximation algorithm for $P|r_j, d_j|\min(m)$ proposed by Cieliebak *et. al.* (2004).

2.3 Fixed-parameter tractability

Finally, we study the fixed-parameter tractability of our problem — i.e., finding parameters k for which there is an algorithm solving instances $\mathcal{I}$ of our problem in $f(k) \cdot poly(|\mathcal{I}|)$ time, where f is a computable function. In this subsection, we suppose that all time values are integers — i.e., day length L, processing times p_j, daily release dates r_j and daily deadlines d_j. By Remark 1, we also assume that all jobs j have processing time at most L.

We consider four parameters: the number of days $\#days$, width μ — i.e., the maximum number of overlapping time windows —, slack σ — i.e., the maximum difference between the length of the daily time window of a job and its processing time — and border flexibility β — i.e., the maximum, over all jobs j, of the minimum between their processing time minus one and their slack plus one. Our problem is para-NP-complete parameterized by each individual parameter (Cieliebak *et. al.* 2004, Hanen and Munier Kordon 2023). We show that our problem is fixed-parameter tractable parameterized by $(\#days + \sigma)$ and $(\mu + \beta)$

by adapting existing fixed-parameter algorithms on the identical parallel machine setting. In both cases, the main obstacle is to find a way to deal with day-overlapping jobs, i.e., jobs which can be processed over two consecutive days.

Theorem 6. $1|periodic_L(r_j, d_j)|C_{\max}$ *is fixed-parameter tractable parameterized by* $(\mu + \beta)$.

Proof. (Sketch.) We adapt the border schedule enumeration algorithm proposed by Tarhan *et. al.* (2023). Given $t \in \mathbb{R}_{>0}$, a border schedule τ at time t is defined as a schedule over a subset of jobs j such that: $\tau(j) < t < \tau(j) + p_j$. Here, we revisit this notion in our periodic setting of period L. We first enumerate over border schedules at daily time $(0 \bmod L)$. Once such a border schedule is fixed, no other job can be processed over multiple days. Thus the rest of the algorithm can unfold in a similar way as in the identical parallel machine setting. This leads to a $\mathcal{O}([2(\mu \cdot \beta + 1)]^{3\mu} \cdot \mu^{\mu+1} \cdot n + n \log(n))$ running time. □

Theorem 7. $1|periodic_L(r_j, d_j)|C_{\max}$ *is fixed-parameter tractable parameterized by* $(\#days + \sigma)$.

Proof. (Sketch.) We adapt the start time enumeration algorithm proposed by Cieliebak *et. al.* (2004) in their section 5.2. In order to deal with day-overlapping jobs, we enumerate over day choices on top of daily start time choices. This leads to a running time in $\mathcal{O}([(\sigma + 1) \cdot \#days]^{(2\sigma+1)\cdot\#days} \cdot n + n \log(n))$. □

3 Discussion

This work gives a first insight on the computational complexity of scheduling low-frequency radio-astronomical observations. Theoretically speaking, we believe that the main remaining open question is whether there is a constant approximation algorithm for our problem. In fact, Theorem 5 relates it to a corresponding longstanding open question about the machine minimization problem (Cieliebak *et. al.* 2004). And, while insightful, current fixed-parameter algorithms are too slow to be used in practice, notably compared to the single-exponential algorithm proposed in Theorem 2.

Going forward, we believe that the model could be refined accordingly to the real-life setting. Because low-frequency radio observations often take several hours to complete, day length L could be reinterpreted as a number of daily time steps, i.e., as an integer, possibly given in unary in the input with a reasonably large time step value (e.g., 15min). Furthermore, since observations can usually be split in chunks, as long as the length of each chunk is long enough, some limited preemption could be introduced to the model in the hope of decreasing the minimum makespan value significantly.

References

Cieliebak, M., Erlebach, T., Hennecke, F., Weber, B. and Widmayer, P., 2004, "Scheduling with release times and deadlines on a minimum number of machines". In: *Exploring New Frontiers of Theoretical Informatics*, pp. 209-222, Springer, Boston MA.

Dereniowski, D. and Kubiak, W., 2010, "Makespan minimization of multi-slot just-in-time scheduling on single and parallel machines", *Journal of Scheduling*, Vol. 13, pp. 479-492.

Hanen, C. and Munier Kordon, A., 2023, "Fixed-parameter tractability of scheduling dependent typed tasks subject to release times and deadlines", *Journal of Scheduling*, Vol. 27, pp. 1-15.

Keil, J., 1992, "On the complexity of scheduling tasks with discrete starting times", *Operations Research Letters*, Vol. 12, pp. 293-295.

Lenstra, J.K., Rinnooy Kan, A.H.G. and Brucker, P., 1977, "Complexity of machine scheduling problems", *Ann. of Discrete Math.*, Vol. 1, pp. 343-362.

Tarhan, I., Carlier, J., Hanen, C., Jouglet, A. and Munier Kordon, A., 2023, "Parameterized Analysis of a Dynamic Programming Algorithm for a Parallel Machine Scheduling Problem". In: *European Conference on Parallel Processing*, pp. 139-153, Springer.

Philippe Zarka, Laurent Denis, Michel Tagger, Julien Girard, Andrée Coffre, et al., 2020, "The low-frequency radio telescope NenuFAR". In: *URSI GASS 2020*, pp. 139-153, Springer.

Moderate Exponential-time Quantum Dynamic Programming Across the Subsets for one machine Scheduling Problems [1]

Camille Grange[1], Michael Poss[2], Eric Bourreau[2], Vincent T'kindt[3], et Olivier Ploton[3]

[1] LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, France
[2] LIRMM, CNRS, University of Montpellier, France
[3] LIFAT, CNRS, University of Tours, France

Abstract. We present a bounded-error hybrid quantum–classical algorithm that improves the worst-case complexity of Dynamic Programming Across the Subsets (DPAS) for a wide class of NP-hard scheduling problems. Building on the Quantum Minimum Finding (QMF) algorithm of Durr and Hoyer, and extending the seminal work of Ambainis et al., we design a unified algorithmic approach capable of handling additive, composed, and decision-type recurrences. As a consequence, we obtain improved exponential factors $O^\star(c^n_{\text{quant}})$ with $c_{\text{quant}} < c_{\text{classical}}$ for several classical scheduling problems, including one machine problems with deadlines, tardiness, release dates.

1 Introduction

Dynamic Programming Across the Subsets (DPAS) is one of the most powerful paradigms for deriving moderate exponential-time algorithms for NP-hard problems. For scheduling, many classical DPAS formulations yield exact algorithms with complexities of the form $O^\star(c^n)$, where c is often close to 2.

Quantum computing offers new opportunities to reduce these exponential factors. Grover's algorithm and its generalization, the Quantum Minimum Finding (QMF) algorithm of Durr and Hoyer allow the replacement of exhaustive minimization over exponentially large sets by quantum procedures of quadratic speed-up. The seminal work of Ambainis exploited this idea for classical DP approaches to combinatorial problems such as TSP in $O^\star(1.728^n)$ and Set Cover in $O^\star(poly(m,n) \cdot 1.728^n)$. Other NP-hard problems have been tackled with this idea and have led to quantum speed-ups for the Steiner Tree problem, the Graph Coloring problem and the Subset Sum problem.

However, many scheduling problems rely on richer structures than those handled in (A. Ambainis 2019). In particular, they involve:

- temporal feasibility constraints (deadlines, release dates, precedences),
- non-linear objectives (tardiness, weighted lateness),
- auxiliary parameters controlling feasibility (e.g., number of late jobs),
- or even composed decision instead of classical additive ones.

This work establishes a *unified* hybrid quantum-classical methodology, requiring more elaborate composition operators, called *Quantum Dichotomic DPAS* (Q-DDPAS), that captures all these cases under a single analytical framework and yields guaranteed improvements in the exponential part of the complexity.

2 DPAS Structures in Scheduling

Let us describe the type of combinatorial optimization problems $\mathcal{P}$ on which DPAS can be applied. An instance of problem $\mathcal{P}$ is denoted by $\mathcal{I}$ and is described by a tuple of vectors of dimension n (the number of elements in the tuple depends on $\mathcal{P}$). We can think as an example of a single-machine scheduling problem with n jobs where the vectors specify processing times, due dates, and deadlines, among others. Any solution to $\mathcal{P}$ is a permutation of $[n] := \{1, \ldots, n\}$. Our nominal optimization problem can be cast as follows:

$$\mathcal{P}(\mathcal{I}): \qquad \min_{\pi \in \Pi(\mathcal{I})} f(\pi, \mathcal{I}), \tag{1}$$

where $\Pi(\mathcal{I}) \subseteq S_{[n]}$ is the set of feasible permutations of $[n]$ according to given constraints and f is the objective function, which both depend on $\mathcal{I}$. As we will solve P by dynamic programming based on a recurrence formula, it is convenient to define sub-instances as follows.

$$P(J,t): \qquad \min_{\pi \in \Pi(J,t)} f(\pi, J, t)\,, \tag{2}$$

where $\Pi(J,t) \subseteq S_J$ is the set of feasible permutations of J according to the given constraints and $f(.,J,t)$ is the objective function. This problem $P(J,t)$, with an auxiliary time parameter $t \in T$, will be solved by Dynamic Programming over a subset $J \subseteq [n]$ (DPAS). We note $\mathrm{OPT}[J,t]$ the optimal value of $P(J,t)$, for $J \subseteq [n]$ and $t \in \mathbb{Z}$. The nominal problem $\mathcal{P}$ schedules all the jobs and starts at time $t = 0$. In other words, we wish to solve $P([n],0)$ and find the optimal value $\mathrm{OPT}[[n],0]$. Let's introduce two DP recurrences families.

2.1 Additive DPAS

Additive DPAS (A-DPAS) applies when the cost of placing a job last in J augments the optimal value of the remaining set by an additive term. There exists a function $g : 2^{[n]} \times [n] \times T \to \mathbb{R}$, computable in polynomial time, such that, for all $J \subseteq [n]$ and for all $t_0 \in T$,

$$\mathrm{OPT}[J,t] = \min_{j \in J} \big\{ \mathrm{OPT}[J \setminus \{j\}, t] + g(J,j,t) \big\}, \tag{Add-DPAS}$$

initialized by $\mathrm{OPT}[\emptyset, t_0] = 0$.

For a given J, the values $\{\mathrm{OPT}[J \setminus \{j\}, 0] : j \in J\}$ are known, so $\mathrm{OPT}[J,0]$ is computed in time $poly(n) \cdot k$ according to Equation (Add-DPAS) (the computation of g is polynomial). The total complexity of computing $\mathrm{OPT}[[n],0]$ is

$$\sum_{k=1}^{n} poly(n) k \binom{n}{k} = poly(n) \cdot n \cdot 2^{n-1} = \mathcal{O}^*(2^n).$$

(Add-DPAS) solves $\mathcal{P}$ in $\mathcal{O}^*(2^n)$.

When constraints preserve feasibility across concatenation of two halves of J, a dichotomic form exists, call Additive Dichotomic DPAS (AD-DPAS). There exist two functions $t_{\text{shift}} : 2^{[n]} \times 2^{[n]} \times T \to T$ and $h : 2^{[n]} \times 2^{[n]} \times T \to \mathbb{R}$, computable in polynomial time, such that, for all $J \subseteq [n]$ of even cardinality, and for all $t \in T$,

$$\mathrm{OPT}[J,t] = \min_{X \subseteq J, |X| = |J|/2} \big\{ \mathrm{OPT}[X,t] + h(J,X,t) + \mathrm{OPT}[J \setminus X, t_{\text{shift}}(J,X,t)] \big\}, \tag{Add-D-DPAS}$$

initialized by the values $\mathrm{OPT}[\{j\},t]$ for each $j \in [n]$ and $t \in T$.

Thus, computing all $\mathrm{OPT}[J,t]$ for any J of size 2^k and $t \in T$ is done in time $|T| poly(n) \binom{2^k}{2^{k-1}} \binom{n}{2^k}$. The total complexity is equal to

$$C(n) = |T| poly(n) \sum_{k=1}^{N} \binom{2^k}{2^{k-1}} \binom{n}{2^k}.$$

A lower bound on $C(n)$ is the sum of the two last terms:

$$C(n) > |T| poly(n) \left(\binom{n}{n/2} + \binom{n}{n/2} \binom{n/2}{n/4} \right) \approx A |T| poly(n) \frac{2^{1.5n}}{n},$$

where A is a constant. The asymptotic equivalent is readily obtained with the Stirling equivalent for factorials, $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$ for $n \in \mathbb{N}$. Thus, C dominates asymptotically $n \mapsto |T| \cdot 2^n$. In other words, $C(n) = \omega(|T| \cdot 2^n)$. (Add-D-DPAS) solves $\mathcal{P}$ in $\omega(|T| \cdot 2^n)$.

Additive DPAS naturally arises in:

- $1|\tilde{d}_j| \sum w_j C_j$ (deadlines),
- $1|| \sum w_j T_j$ (tardiness),
- $1|\mathrm{prec}| \sum w_j C_j$ (precedence constraints).

2.2 Composed DPAS

Some strongly NP-hard problems cannot be decomposed additively. Instead, feasible schedules must satisfy constraints by composition. The recurrence formulas derived from the work of Lawler for the problem $1|r_j, pmtn|\sum w_jU_j$, where $\epsilon = \sum_{j\in J} w_jU_j$. Let's define, for all $J \subseteq [n]$, $t \in T$ and $\epsilon \in E$,

$$\mathrm{OPT}[J,t,\epsilon] = \min_{\substack{\epsilon' \in E \\ j\in J}} \Big\{ \mathrm{OPT}\Big[\{j\}, \mathrm{OPT}[J\setminus\{j\}, t, \epsilon-\epsilon'], \epsilon'\Big]\Big\}, \tag{Comp-DPAS}$$

initialized by the values of $\mathrm{OPT}[\{j\},t,\epsilon]$ for all $j \in [n]$, $\epsilon \in E$ and $t \in T$. Notice that for $J \subseteq [n]$, $t \in T$ and $\epsilon \in E$, we adopt the convention $\mathrm{OPT}[J,t,\epsilon] = +\infty$ for $\epsilon \notin E$. Let $\epsilon_0 \in E$.

DP solves $P'([n],0,\epsilon_0)$ in $\mathcal{O}^*(|E|^2 \cdot |T| \cdot 2^n)$ where for a given $\epsilon_0 \in E$, the optimal value of $P'(J,t,\epsilon_0)$ is the minimum value of all possible composition of optimal values of the problem on sub-instances with parameters ϵ_1 and ϵ_2 such that $\epsilon_1 + \epsilon_2 = \epsilon_0$. Then (Comp-DPAS) solves $\mathcal{P}$ in $\mathcal{O}^*(|E|^3 \cdot |T| \cdot 2^n)$.

A dichotomic variant also exists. For all $J \subseteq [n]$ of even cardinality, $t \in T$ and $\epsilon \in E$,

$$\mathrm{OPT}[J,t,\epsilon] = \min_{\substack{\epsilon' \in E \\ X\subset J : |X| = |J|/2}} \Big\{ \mathrm{OPT}\Big[X, \mathrm{OPT}[J\setminus X, t, \epsilon-\epsilon'], \epsilon'\Big]\Big\}, \tag{Comp-D-DPAS}$$

initialized by the values of $\mathrm{OPT}[\{j\},t,\epsilon]$ for all $j \in [n]$, $t \in T$ and $\epsilon \in E$. Let $t_0 \in T$ and $\epsilon_0 \in E$.

(Comp-D-DPAS) solves $\mathcal{P}$ in $\omega(|E|^3 \cdot |T| \cdot 2^n)$.

Composed DPAS applies to problems with release dates and non-linear cumulative costs, e.g.:

- $1|r_j|\sum w_jU_j$ (late jobs)
- $1|r_j|\sum w_jC_j$ (weighted completion times with release dates)

3 Hybrid Algorithm Q-DDPAS

Q-DDPAS generalizes Ambainis framework while supporting all DPAS types described above. The algorithm proceeds in two main phases.

3.1 Phase 1: Classical Preprocessing

All subproblems of size up to $n/4$ are solved using $X \subseteq [n] : |X| \leq n/4$ and $t \in T$ where we compute the optimal value $\mathrm{OPT}[X,t]$ and the corresponding permutation $\pi^*[X,t]$ by classical corresponding DPAS and store the tuple $(X, t, \mathrm{OPT}[X,t], \pi^*[X,t])$ in the QRAM that serve as the base of the quantum evaluation oracles.

Solving all $\mathrm{OPT}[X,t]$ for all X of size $n/4$ and for all $t \in T$ with (Add-DPAS) is in time

$$|T|poly(n)\sum_{k=1}^{n/4} k\binom{n}{k} = \mathcal{O}^*\left(|T|\binom{n}{\leq n/4}\right).$$

Thus, because $\mathcal{O}^*\left(\binom{n}{\leq n/4}\right) = \mathcal{O}^*(2^{0.811n})$, the complexity of the classical part is $\mathcal{O}^*(|T| \cdot 2^{0.811n})$.

3.2 Phase 2: Quantum Dichotomic Minimization

For subsets J of sizes $n/2$ and n, the DPAS recurrence requires minimizing over all balanced partitions $X \subseteq J$. Classically, this requires $O\Big(\binom{|J|}{|J|/2}\Big)$ but using QMF, this is reduced to $O\Big(\sqrt{\binom{|J|}{|J|/2}}\Big) \approx O\Big(\frac{2^{|J|/2}}{|J|^{1/4}}\Big)$, yielding an improved exponential factor.

We apply a second time the dichotomic search over all $O\Big(\binom{|J|/2}{|J|/4}\Big)$. To compute complexity, in 2.1, it was shown that Add-D-DPAS over only $n/4$ was $\mathcal{O}^*(2^{1.5n})$, this dominates the first part (explaining why this kind of decomposition is never used), but applying QMF allow complexity of $\mathcal{O}^*(2^{\sqrt{(1.5)}n}) = \mathcal{O}^*(2^{0.75n})$.

Total complexity of Q-DDPAS is

$$\mathcal{O}^*\left(2^{0.75n} + |T| \cdot 2^{0.811n}\right) = \mathcal{O}^*\left(|T| \cdot 2^{0.811n}\right) = \mathcal{O}^*\left(|T| \cdot 1.754^n\right) .$$

A slight modification of Q-DDPAS amounts add a third level of recurrence in the quantum part, but instead of searching for the best concatenation among all the bi-partition of size $(n/8, n/8)$ (i.e. solving Equation (Add-D-DPAS) for $|J| = n/4$), we search for the best concatenation among all the bi-partitions of size $(0.945 \cdot \frac{n}{4}, 0.055 \cdot \frac{n}{4})$, then the complexity is $\mathcal{O}^*(|T| \cdot 2^{0.789n})$ equally balance between classical and quantum part.

The same reasonings applies to the composed cases.

4 Complexity Bounds

Table 1, and his extension to 3-Flowshop, can be found in (Grange 2025) and summarizes the improvements obtained by Q-DDPAS. In all cases, the exponential constant decreases from $c_{\text{classical}}$ to c_{quant} up to polynomial factors, yielding the numerical value 1.728 for QDDPAS structures.

Problem	Classical	Hybrid Q-DDPAS
$1\|\tilde{d}_j\|\sum w_jC_j$	$O^\star(2^n)$	$O^\star(p([n])\,1.728^n)$
$1\|\|\sum w_jT_j$	$O^\star(2^n)$	$O^\star(p([n])\,1.728^n)$
$1\|\text{prec}\|\sum w_jC_j$	$O^\star((2-\varepsilon)^n)$	$O^\star(1.728^n)$
$1\|r_j\|\sum w_jU_j$	$O^\star(\|E\|\|T\|2^n)$	$O^\star(\|E\|^3\|T\|\,1.728^n)$
$1\|r_j\|\sum w_jC_j$	$O^\star(\|E\|\|T\|^2 2^n)$	$O^\star(\|E\|^3\|T\|^4\,1.728^n)$

Table 1. Exponential improvements obtained by Q-DDPAS (pseudo-polynomial factors omitted).

5 Conclusion

We developed a unified and significantly more general framework for hybrid quantum–classical dynamic programming. Q-DDPAS covers all known DPAS structures in scheduling, including additive, composed, and decision-type recurrences; it strictly generalizes existing quantum DPAS algorithms; and it offers the best asymptotic worst-case complexities for a wide class of NP-hard scheduling problems.

This work demonstrates that quantum speed-ups in exact algorithms are not restricted to isolated combinatorial problems but can be systematically derived wherever DPAS recurrences exist. Future research includes minimizing pseudo-polynomial factors, extending the framework to multi-machine environments with richer precedence structures, and investigating quantum variants of approximation schemes for scheduling.

References

A. Ambainis et al., 2019, "Quantum Speedups for Exponential-Time DP Algorithms", *SODA*

C. Grange, M. Poss, E. Bourreau, V. T'kindt and O. Ploton, 2025, "Moderate exponential-time quantum dynamic programming across the subsets for scheduling problems", *Eur. J. Oper. Res.*, Vol. 01, pp. 5-15.

A New Parameterized Complexity Analysis for Scheduling Precedence-Constrained Tasks on Parallel Machines

Claire Hanen[1,2], Alix Munier-Kordon[1], Istenc Tarhan[3]

[1] LIP6, CNRS, Sorbonne University, Paris and
Claire.Hanen@lip6.fr,Alix.Munier@lip6.fr
[2] Paris Nanterre University
[3] University College, Dublin
istenc.tarhan@ucd.ie

Keywords: Parameterized complexity, scheduling, degeneracy, parallel machines.

1 Introduction

Scheduling tasks on parallel processors with precedence constraints is core problem of scheduling theory, for which complexity results and approximation algorithm have been designed till the seventies[Lenstra and Rinnooy Kan, 1978]. In the last decade, the parameterized complexity provided a new insight on the difficulty of scheduling problems.

Given a parameter k, a problem is called *fixed-parameter tractable* (FPT in short) parameterized by k if any of its instances I can be solved in time $\mathcal{O}(f(k) \cdot poly(|I|))$ with f an arbitrary computable function [Downey and Fellows, 1999]. Such a problem is considered thus tractable if the parameter is bounded. When the studied problem is believed to not be FPT, many complexity classes are available as parameterized analogues to NP like para-NP or the W-hierarchy [Flum and Grohe, 1998].

Parameterized complexity theory of scheduling problems has been recently widely studied for problems with time windows, particularly by considering parameters linked to their structure. Let us mention the parameters pathwidth [Munier-Kordon, 2021],[Hanen and Munier-Kordon, 2024], [Tarhan et al., 2025], slack [van Bevern et al., 2016] [Cieliebak et al., 2004], [van Bevern et al., 2017] and proper level [Mallem et al., 2024]. For problems with task rejection, some authors [Heeger and Hermelin, 2024] developed parameterized complexity analysis based on the number of different values (weights, processing times, deadlines, release times).

If only precedence constraints are considered, and for the makespan objective, authors have considered parameters such as the number of machines or the width of the precedence graph. Negative results have been provided, even for unit processing time tasks. For example in [Bodlaender et al., 2022], the authors showed that $P|prec, p_j = 1|C_{max}$ is XNLP-complete when the number of machines is paired with precedence width.

In this paper, we focus on the degeneracy that depends solely on the structure of the precedence graph. It is larger than the width and, when time windows

induced by a makespan bound are taken into account, it becomes independent of the pathwidth. A recent result [Munier-Kordon and Hanen, 2025] established that minimizing the makespan for precedence-constrained tasks of unit processing times is FPT when parameterized by degeneracy. We show here that this result extends to arbitrary processing times and to several objective functions (makespan, lateness, total weighted completion time), when parameterized by both the degeneracy and the maximum processing time.

2 Problem and parameter definition

Let us consider a set $\mathcal{T}$ of tasks, m parallel processors and a precedence directed acyclic graph $\mathcal{G}(\mathcal{T}, \mathcal{A})$. A task j is a successor (resp. predecessor) of i if there is a path from i to j (resp. from j to i) in $\mathcal{G}(\mathcal{T}, \mathcal{A})$. We also assume a dummy source and dummy sink node in $\mathcal{G}(\mathcal{T}, \mathcal{A})$.

Each task $i \in \mathcal{T}$ is characterized by its processing time $p_i \in \mathbb{N} - \{0\}$. We denote by p_{max} the maximum processing time of a task. A schedule assigns to each task i a completion time C_i such that for any arc $(i, j) \in \mathcal{A}$, $C_i + p_i \leq C_j$ and no more than m tasks are in progress at the same time. We also consider several optimization criteria, namely the makespan $C_{max} = \max_{i \in \mathcal{T}} C_i$, the lateness $L_{max} = \max_{i \in \mathcal{T}} C_i - d_i$ if due dates are given, and the weighted completion time $\sum_{i \in \mathcal{T}} w_i C_i$.

We build the co-comparability graph $H_{\mathcal{G}(\mathcal{T},\mathcal{A})} = (\mathcal{T}, E)$ induced by $\mathcal{G}(\mathcal{T}, \mathcal{A})$ as follows. Two tasks $(i, j) \in \mathcal{T}^2$ are linked by an edge $e = ij \in E$ if there is no path from i to j or from j to i in $\mathcal{G}(\mathcal{T}, \mathcal{A})$. The degeneracy of the co-comparability graph $\delta = d(H_{\mathcal{G}(\mathcal{T},\mathcal{A})})$ is the minimum number k such that in any subset $S \subseteq \mathcal{T}$ there is at least a task $i \in S$ with degree at most k in the subgraph of $H_{\mathcal{G}(\mathcal{T},\mathcal{A})}$ induced by S. The degeneracy in this case can be computed easily in polynomial time [Lick and White, 1970].

3 Fixed parameter algorithms

Observe that as the problem $P|prec, p_j = 1|C_{max}$ [Ullman, 1975] is strongly NP-hard and the same complexity holds for the weighted completion time objective too [Lenstra and Rinnooy Kan, 1978], our problems are para-NP-hard for parameter p_{max}. However in this paper we prove the following result (proofs are omitted due to the lack of space):

Theorem 1. *Scheduling problems $P|prec|C$ for $C \in \{C_{max}, L_{max}, \sum_{i \in \mathcal{T}} w_i C_i\}$ are FPT parameterized by both δ and p_{max}.*

Let us consider a feasible schedule. At any time t, at most m tasks are in progress, and all tasks started before t have all their predecessors already scheduled. Let us denote by B the border at time t, i.e. the set of tasks started before t none of whose successors have started before t. Then, we observe that the set $C(B)$ of all tasks started before t equals the set of predecessors of B plus

B. Moreover, all tasks in progress at t are in B, and as tasks in B do not have precedence relations, B defines a clique in $H_{\mathcal{G}(\mathcal{T},\mathcal{A})}$.

Let us consider now a feasible schedule and the time t corresponding to the completion time of a task. Then we can characterize the partial schedule of tasks scheduled before t by a border B and the profile of the tasks in progress at t, specifically their name and remaining processing time after t. This leads us to the following definition of the state graph:

A **State** is a tuple $\sigma = (B, S)$ where B is a clique of $H_{\mathcal{G}(\mathcal{T},\mathcal{A})}$, and $S : B \to \mathbb{N}$. For any task $i \in B$, $S(i) = C_i - t > 0$ if i is not completed at time t, otherwise $S(i) = 0$. Note that there is at most $m - 1$ tasks i for which $S(i) > 0$. More generally, a state is associated with all partial schedules of $C(B)$ that if t is the maximum completion time of a task of $i \in B$ with $S(i) = 0$ then the completion time of any task $i \in B$ is $C_i = t + S(i)$ if $S(i) > 0$ or $C_i \leq t$ otherwise.

The core result that will lead to a FPT complexity is based on the fact that the number of cliques of $H_{\mathcal{G}(\mathcal{T},\mathcal{A})}$ is FPT parameterized by the degeneracy δ [Munier-Kordon and Hanen, 2025].

Lemma 1. *The number of states is bounded by* $n2^{\delta} \cdot p_{max}^{\delta}$.

The **Arcs** $a = (\sigma, \sigma')$ of the state graph express that at least one partial feasible schedule associated to $\sigma' = (B', S')$ may be built from $\sigma = (B, S)$ by scheduling tasks in $B' \backslash B$ at time t. More precisely, to extend a state σ, a subset X of at most $m - |\{i \in B, S(i) > 0\}|$ tasks should be selected among feasible tasks, i.e. whose predecessors are in $C(B)$ and have no predecessor with $S(i) > 0$. For each subset X, we schedule them on the free processors at the time t and update it to the next completion time of task in $B \cup X$. The resulting border B' and associated S' function can then be easily computed.

Dynamic programming algorithms The principle of FPT algorithms for our three different objective functions consists in generating the state graph using a memoization mechanism for the states. When extending a state σ to a state σ', the optimal path to σ plus the arc (σ, σ') defines a path that can be evaluated according to the objective function. Then each state $\sigma = (B, S)$ is evaluated by the minimum value of the objective function for partial schedules associated with the state. We can then use Bellman's shortest path algorithm on directed acyclic graphs to compute the optimal path from the source node to the sink of the state graph. Its complexity only depends on the number of states.

4 Conclusion

In this paper we provided a FPT algorithm for parameter degeneracy combined with maximum processing time for scheduling tasks with precedence constraints on parallel machines. The algorithm can handle different objective functions. The parameterized complexity of our problems for only degeneracy is still open.

Another important perspective of this work is the implementation of the dynamic programming scheme, using bounds to limit the practical complexity and comparing the performance with other exact algorithms.

References

Bodlaender et al., 2022. Bodlaender, H. L., Groenland, C., Nederlof, J., and Swennenhuis, C. M. (2022). Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 193–204. IEEE.

Cieliebak et al., 2004. Cieliebak, M., Erlebach, T., Hennecke, F., Weber, B., and Widmayer, P. (2004). Scheduling with release times and deadlines on a minimum number of machines. In Levy, J.-J., Mayr, E. W., and Mitchell, J. C., editors, *Exploring New Frontiers of Theoretical Informatics*, pages 209–222, Boston, MA. Springer US.

Downey and Fellows, 1999. Downey, R. and Fellows, M. (1999). *Parameterized Complexity*. Springer.

Flum and Grohe, 1998. Flum, J. and Grohe, M. (1998). *Parameterized Complexity Theory*. Springer.

Hanen and Munier-Kordon, 2024. Hanen, C. and Munier-Kordon, A. (2024). Fixed-parameter tractability of scheduling dependent typed tasks subject to release times and deadlines. *J. Sched.*, 27(2):119–133.

Heeger and Hermelin, 2024. Heeger, K. and Hermelin, D. (2024). Minimizing the weighted number of tardy jobs is W[1]-hard. *arXiv preprint arXiv:2401.01740*.

Lenstra and Rinnooy Kan, 1978. Lenstra, J. and Rinnooy Kan, A. (1978). Complexity of scheduling under precedence constraints. *Oper. Res.*, 26(1):22–35.

Lick and White, 1970. Lick, D. R. and White, A. T. (1970). k-degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096.

Mallem et al., 2024. Mallem, M., Hanen, C., and Munier-Kordon, A. (2024). A new structural parameter on single machine scheduling with release dates and deadlines. In *ISCO 2024*, volume 14594 of *Lecture Notes in Computer Science*, pages 205–219. Springer.

Munier-Kordon, 2021. Munier-Kordon, A. (2021). A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. *Discret. Appl. Math.*, 290:1–6.

Munier-Kordon and Hanen, 2025. Munier-Kordon, A. and Hanen, C. (2025). Exploiting structural degeneracy for fpt results on problems over directed acyclic graphs. *Manuscript submitted for publication*.

Tarhan et al., 2025. Tarhan, I., Hanen, C., Munier-Kordon, A., Carlier, J., and Jouglet, A. (2025). FPT implicit enumeration of active schedules. *Discret. Appl. Math.*, 376:235–250.

Ullman, 1975. Ullman, J. D. (1975). NP-complete scheduling problems. *Journal of Computer and System sciences*.

van Bevern et al., 2016. van Bevern, R., Bredereck, R., Bulteau, L., Komusiewicz, C., Talmon, N., and Woeginger, G. J. (2016). Precedence-constrained scheduling problems parameterized by partial order width. In *Discrete Optimization and Operations Research*, pages 105–120, Cham. Springer International Publishing.

van Bevern et al., 2017. van Bevern, R., Niedermeier, R., and Suchỳ, O. (2017). A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *Journal of Scheduling*, 20:255–265.

H - Constraint programming #2

A Constraint Programming Model for the Cyclic Aerospace Line-Balancing Problem

Diego Olivier Fernandez Pons[1] and Pierre Schaus[2]

[1] OptalCP, France
diego.olivier.fernandez.pons@optalcp.com
[2] UCLouvain, Louvain-La-Neuve, Belgium
pierre.schaus@uclouvain.be

Keywords: Cyclic scheduling, line-balancing, RCPSP, Constraint Programming.

1 An aerospace line-balancing problem with cyclic schedules

The aerospace line-balancing problem consists of assigning a set of tasks to a set of workstations in order to assemble aircraft, while determining a schedule for the tasks at each workstation. The objective is to ensure a common minimum cycle time at every station, while satisfying precedence relations and shared resource constraints across stations. We propose a new constraint programming model for a simplified version of the problem without parallel stations. The model avoids the use of modulo operators to enforce cyclic resource constraints, resulting in a formulation that is both simpler and more computationally efficient. Preliminary experiments demonstrate the superiority of the proposed approach and highlight the efficiency of OptalCP in solving this model compared to CP Optimizer.

2 Line Balancing Models

The line-balancing problem considered in this work comes from (Pucel 2024) and consists of assigning a set of tasks to a set of stations and scheduling them so as to minimize the cycle time c, while satisfying precedence constraints and shared resource capacity constraints. The resource capacity constraint applies to each cycle $[0, c]$ for all stations simultaneously. It results from the assembly-line having walking workers that can walk in the same cycle from station to station to perform tasks.

We present two constraint programming formulations of this problem that differ only in how precedence and capacity constraints are expressed.

2.1 Temporal Model

The temporal formulation is a simplified version of the model introduced in (Pucel 2024) for the more complex problem with parallel stations. In this model each task O^α is represented in absolute time. Assignment of tasks to stations is captured by an integer variable $s^\alpha \in [0, \text{max\_stations} - 1]$. If assigned to station s, O^α must lie in the interval $[s \times c, (s + 1) \times c]$.

$$\min c$$

$$\forall(\alpha,\beta) \in \text{precedences } \texttt{end}(O^\alpha) \leq \texttt{start}(O^\beta) \tag{1}$$
$$\forall\alpha \ \texttt{start}(O^\alpha) = \texttt{start}(M^\alpha) + s^\alpha \times c \tag{2}$$
$$\forall\alpha \ \texttt{end}(O^\alpha) = \texttt{end}(M^\alpha) + s^\alpha \times c \tag{3}$$
$$\forall r \ \sum_\alpha \text{consumption}_r^\alpha \times \texttt{pulse}(M^\alpha) \leq \text{capacity}_r \tag{4}$$

Constraint (1) enforces precedences on absolute intervals. Constraints (2) and (3) introduce, for each task O^α, a projected interval M^α defined inside the cycle window $[0, c]$. Formally, if $O^\alpha = [\texttt{start}(O^\alpha),\ \texttt{end}(O^\alpha)]$, then M^α is the interval $M^\alpha = \big[\texttt{start}(O^\alpha) \bmod c,\ \texttt{end}(O^\alpha) \bmod c\big]$. It represents the projection of O^α into the cycle window and is used to express resource capacity constraints within that window. Resource capacities (4) are enforced on these projected intervals. In this formulation, precedences act on absolute intervals, whereas capacities act on projected intervals obtained through a modulo transformation.

2.2 Spatial Model

We propose a new spatial formulation, where all task intervals are directly defined within the cycle window $[0, c]$.

$$\min c$$

$$\forall\alpha \ \texttt{end}(M^\alpha) \leq c \tag{5}$$
$$\forall(\alpha,\beta) \in \text{precedences},\ s^\alpha \leq s^\beta \tag{6}$$
$$\forall(\alpha,\beta) \in \text{precedences},\ s^\alpha = s^\beta \Rightarrow \texttt{end}(M^\alpha) \leq \texttt{start}(M^\beta) \tag{7}$$
$$\forall r \ \sum_\alpha \text{consumption}_r^\alpha \times \texttt{pulse}(M^\alpha) \leq \text{capacity}_r \tag{8}$$

Constraint (6) ensures all projections M^α remain in the cycle $[0, c]$. Precedences are enforced directly on these intervals through station-order consistency constraints (6) and (7). Resource capacities (8) are also expressed directly on these intervals.

2.3 Structural Difference

The fundamental distinction between the two formulations lies in the trade-off between the complexity of modeling precedence relations versus resource constraints, and the resulting impact on constraint propagation.

In the *Temporal Model*, decision variables represent the absolute execution time of tasks across the entire assembly horizon. While this representation simplifies the expression of precedence constraints (Eq. (1)), it necessitates the projection of task intervals into the cycle window $[0, c]$ via a modulo operator or equivalently with arithmetic constraints (Eq. (2)–(3)) to enforce resource capacities. From a constraint programming perspective, the modulo operator acts as a barrier to propagation. Domain reductions on the projected interval M^α (derived from resource contention) do not efficiently propagate back to the absolute interval O^α when the domain of O^α spans multiple cycles. Consequently, the solver struggles to prune the search space effectively, as it cannot easily deduce that a task is forbidden at a specific absolute time based solely on resource usage within the cycle.

Conversely, the *Spatial Model* defines decision variables directly within the cycle window $[0, c]$, augmented by station assignment variables. This formulation eliminates the modulo operator entirely, allowing the global filtering algorithms of the `cumulative` and `noOverlap` constraints to operate directly on the primary decision variables. The complexity is instead shifted to the precedence constraints (Eq. (6)–(7)), which must now account for station indices.

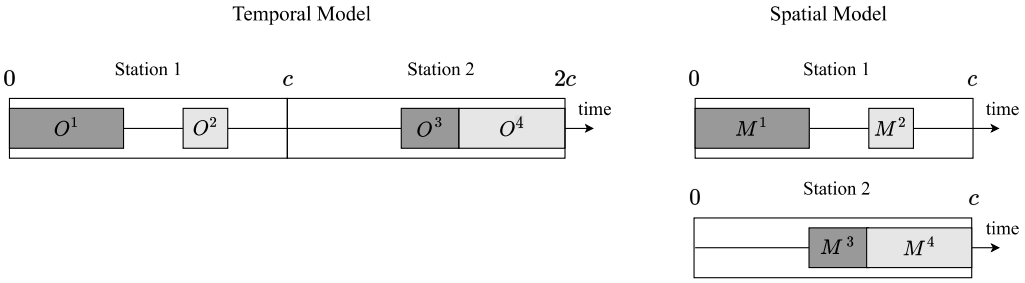


Fig. 1. Illustrative example of the difference between the *Temporal Model* and the *Spatial Model*.

Example 1. Consider the instance depicted in Figure 1, involving four tasks and two stations. Task 1 must precede Task 4 ($1 \to 4$), and both tasks require the same disjunctive resource (indicated by the same color). In the *Temporal Model*, tasks are scheduled on the absolute horizon $[0, 2c]$ while they are scheduled on the horizon $[0, c]$ for the *Spatial Model*. The precedence constraint is satisfied structurally because the station index of Task 1 is strictly lower than that of Task 4. Since the assembly process imposes a sequential flow of products through the stations, this assignment guarantees temporal precedence regardless of the tasks' local offsets within the cycle.

2.4 Parallel Alternative Stations

The problem introduced in (Pucel 2024) is slightly more general than the formulations presented above, as it allows for alternative parallel workstations.

For the sake of simplicity and conciseness, this extension is not detailed here. However, both the spatial and the temporal models can be naturally extended to accommodate parallel alternative stations by introducing additional optional intervals per station alternative.

3 Experimental Evaluation

Previous results

Pucel and Roussel (Pucel 2024) results are shown on table 1. They used the extended temporal formulation with between one and three stages. At most one stage could contain parallel alternative stations. And they tested with the first 150 instances of the j30rcp PSPlib benchmark on a 20-core Intel Xeon CPU @2.60Ghz with 62GB of RAM.

Our results

Our results shown in table 2 are obtained on an i7 4-core @3.3GHz with 32GB of RAM.

Table 1. Subset of the results of Pucel and Roussel that don't use parallel stations

Layout	Solved	Optimal	Avg. Time (s)	engine
[1]	150	150	2.5	CP Optimizer
[1,1]	150	122	72.3	CP Optimizer
[1,1,1]	150	4	151.1	CP Optimizer

With layout [1] the problem is just an RCPSP and what is being measured is overhead caused by the models when the line balancing dimension is completely fixed. Therefore we report the results with an rcpsp model as well.

Both the temporal and the spatial model perform significantly better in our results, with the caveat that the model from (Pucel 2024) is capable of solving problems with parallel stations, and ours aren't (in their non-extended form presented here).

While the difference between temporal and spatial model seems modest in average, the spatial model finds equal or better solutions in all instances but 3, and when the solution found is of similar value, the spatial model finds it 2.1x faster in average.

Table 2. Our results

Model	Layout	Solved	Optimal	Avg. Time (s)	engine
rcpsp	-	150	150	0.44	OptalCP
temporal	[1]	150	150	0.50	OptalCP
spatial	[1]	150	150	0.49	OptalCP
temporal	[1,1,1]	150	113	91	OptalCP
spatial	[1,1,1]	150	116	80	OptalCP

The comparison between RCPSP and line-balancing shows the presence of stations significantly degrades the interaction between resource and precedence constraints. While the spatial model is able to bridge part of that gap, there is still a significant loss of propagation, which translates into extremely long optimality proofs due to late failures.

Comparision OptalCP vs CP Optimizer

To evaluate how much of the gap in the results was due to the engine used, we requested the 2880 models Pucel and Roussel had used in `.cpo` format and solved them with both CP Optimizer and OptalCP on the same machine (`http://dev.vilim.eu:8000/~petr/Onera/cpo-vs-optal/main.html`)

Table 3. Engine comparison on same models and hardware

	OptalCP	CPO
strictly better solution	776	7
better time with same solution	1869	105

References

Pucel, X. and Roussel, S., 2024, September. Constraint Programming Model for Assembly Line Balancing and Scheduling with Walking Workers and Parallel Stations. In CP 2024.

Preemptive jobshop scheduling with *maximum* workload constraints

Tanguy TERRIEN[1], Cyrille BRIAND[1]

LAAS-CNRS, Universite de Toulouse, France
{tterrien, briand}@laas.fr

Keywords: Constraint Programming, Jobshop scheduling, Workload constraint.

1 Introduction

Optimizing schedules in real-world settings requires accounting for workload constraints, particularly for human resources, in order to ensure regulatory compliance, prevent fatigue, and maintain productivity (Pinedo 2012). Ignoring such constraints may lead to legal penalties, reduced efficiency, or high employee turnover. In contrast, enforcing adequate rest periods and balanced workloads improves worker well-being and overall operational safety. These issues arise in many application domains, such as project management, healthcare (e.g., nurse rostering), and transportation (e.g., train driver scheduling), where regulatory workload rules significantly complicate scheduling decisions (Pinedo 2012).

Constraint Programming (CP) has proved to be an effective paradigm for solving industrial-scale scheduling problems involving complex and heterogeneous constraints (Laborie *et al.* 2018). However, modeling preemption remains challenging, highly increasing combinatorial complexity. To tackle this, we introduce a new class of constraints, called **MaxW constraints** (**Max**imum **W**orkload), which capture rest-time and workload requirements for operators over given time intervals. These constraints provide a compact and expressive way to model regulations such as minimum rest periods or bounds on cumulative workload, while avoiding a decomposition of activities into unit-duration tasks.

In this paper, we focus on the NP-hard Job Shop Scheduling Problem (JSP), which involves sequencing operations on machines. Few works address the integration of workload-constrained operators. Mauguière *et al.* (2005) use branch-and-bound for operator unavailability, while Müller and Kress (2022) propose a filter-and-fan heuristic for flexible JSP under workforce constraints. The Preemptive JSP (pJSP) allows task interruptions, which improves the makespan but increases combinatorial complexity. pJSP with workload constraints further couples personnel and job shop resources, retaining NP-hardness. Related work in online printing shop scheduling uses CP models for resumable operations and machine unavailability (Lunardi *et al.* 2020), providing modeling insights. More recently, Juvin *et al.* (2023) proposed an efficient CP approach for pJSP avoiding full combinatorial enumeration, forming a strong basis for extending to workload-constrained operators.

The main contributions of this paper are as follows: (i) we formalize MaxW constraints for human operators, generalizing workload and rest-time regulations; (ii) we propose a CP model for the preemptive Jobshop Scheduling Problem with MaxW constraints (MaxW-pJSP), implemented using the Mistral solver; (iii) we implement a comparative CP model of the MaxW-pJSP using state-of-the-art IBM® CP Optimizer (CPO) (Laborie *et al.* 2018); (iv) we produce a new MaxW-pJSP benchmark of pJSP instances with added MaxW constraints.

The remainder of the paper is organized as follows. Section 2 presents the definition of the problem and introduces some CP models that solve it. Section 3 reports the experimental evaluation. Section 4 concludes the paper.

2 Problem definition and CP models

A **MaxW** constraint associated with operator k is defined as a triplet $(\delta, [u, v])$, where δ expresses the maximum number of work days that operator k should have within $[u, v]$.

The MaxW-pJSP consists of a set of jobs $\mathcal{J}$, where each job $J_i \in \mathcal{J}$ is composed of an ordered sequence of n_i tasks $t_{i,j}$, $j \in \{1, \ldots, n_i\}$. A set of operators {0, ..., K-1} are available. Each task $t_{i,j}$ is characterized by a processing time $p_{i,j}$, an earliest start time $s_{i,j}$, and a latest completion time $e_{i,j}$, and is assigned to exactly one operator. An operator may interrupt the execution of a task and resume it later, provided that the total executed processing time equals $p_{i,j}$. To regulate operator workload, each operator k is subject to a set of MaxW constraints of the form $(\delta, [u, v])$. These constraints are generated to reflect realistic production regulations (see Section 3 for details).

The objective is makespan minimization. Each task must be fully executed by its assigned operator; tasks within the same job must satisfy precedence constraints. For each operator, working periods and rest periods must not overlap. The MaxW-pJSP extends the classical pJSP by incorporating the MaxW constraints described above.

Mistral model: To improve efficiency, we avoid modeling individual shifts and rely on Mistral's PreemptiveNoOverlap constraint (Juvin *et al.* 2023), which fixes start and end times so that each operator can complete its tasks preemptively. Combined with dedicated propagation (including overload checks), this efficiently solves the pJSP with makespan minimization. A feasible schedule can then be constructed in polynomial time using Jackson's algorithm We extend this framework to model the MaxW-pJSP. Unless otherwise stated, indices range over $i \in \{0, \ldots, |\mathcal{J}| - 1\}$, $j \in \{0, \ldots, n_i\}$, and $k \in$ Operators. Each MaxW constraint is decomposed into a set of non-overlapping subintervals indexed by q (illustrated in Figure 1). We denote by MaxW_k the set of MaxW constraints associated with operator k, and by T_k the set of tasks assigned to operator k. For each MaxW constraint c, the parameters δ_c, u_c, and v_c denote its defining triplet.

$$\min c_{max}$$

$$s_{i,j} \in [0, UB - \text{P}_{i,j}],\ e_{i,j} \in [\text{P}_{i,j}, UB] \qquad \forall i, j \quad \text{(V1)}$$

$$c_{max} \in [0, UB] \qquad \text{(V2)}$$

$$\text{d}_q^k \in [0, |q|] \qquad \forall k, \forall q \quad \text{(V3)}$$

$$e_{i,j} \geq s_{i,j} + \text{P}_{i,j} \qquad \forall i, \forall j \quad \text{(C1)}$$

$$s_{i,j+1} \geq e_{i,j} \qquad \forall i, \forall j \quad \text{(C2)}$$

$$s_{i,0} = 0 \quad \text{and} \quad e_{i,n_i} = c_{max} \qquad \forall i \quad \text{(C3)}$$

$$\textsc{Pre.NoOverlap}\Big(\big\{(s_{i,j}, e_{i,j}, \text{P}_{i,j})\big\}_{\forall i, \forall j \in \text{T}_k} \cup \big\{(S_q, E_q, \text{d}_q^k)\big\}_{\forall q}\Big) \qquad \forall k \quad \text{(C4)}$$

$$v_c - u_c - \sum_{\forall q \in [u_c, v_c]} \text{d}_q^k \leq \delta_c \qquad \forall k, \forall c \in \text{MaxW} \quad \text{(C5)}$$

For each task $t_{i,j}$, we define start and end time variables $s_{i,j}$ and $e_{i,j}$ (V1) (UB is an upper bound on the makespan). We want to minimize the makespan (V2). To model MaxW constraints, the scheduling horizon is partitioned into a finite set of non-overlapping subintervals, whose boundaries are induced by the start and end points of all MaxW constraints. For each operator k and subinterval q, we introduce a variable d_q^k representing the amount of rest allocated to operator k during q, bounded by the length of the subinterval (V3). As an example, a single operator with two MaxW constraints (2,5,0,6) and (3,4,4,9) is illustrated in Figure 1 showing subintervals, duration variables, and constraints.

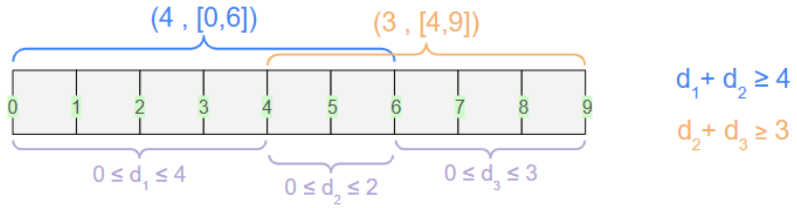


Fig. 1: Two MaxW constraints, their subintervals and associated duration variables

(C1) enforces full execution of tasks, (C2) handles job precedence, and (C3) links job completion to the makespan. For each operator, PreemptiveNoOverlap (C4) prevents overlap between work and rest periods, allowing preemption execution. MaxW constraints (C5) ensure that rest within each interval lies between desired lower and upper bounds. We extended Mistral's propagation to support variable task durations d_q^k, guaranteeing a feasible preemptive schedule with optimal makespan.

CPO model: In CPO, with preemption, each task must be split into multiple unit-length optional intervals, dramatically increasing combinatorial complexity and making propagation more difficult. The proposed explicit CPO model is the following (note that CPO allows for interval variables to be optional). This model was developed using insights from previous work (Lunardi *et al.* 2020). The model uses two families of decision variables. For each task j of activity i, we define unit-length interval variables (V1) corresponding to its processing time. To handle MaxW constraints, each operator k receives δ_c optional unit intervals per MaxW constraint c (V2), which must lie within $[u_c, v_c]$ (C1). These variables provide flexibility for scheduling rest periods. The objective remains makespan minimization. Sequential execution of task units is enforced (C2), along with precedence between successive operations (C3). A no-overlap constraint ensures an operator's work (W) and rest (O) intervals do not overlap (C4). Finally, MaxW constraints require that for each c, at least δ_c optional intervals are scheduled within $[u_c, v_c]$ (C5).

$$\min \quad \max_{i,j} \left(\text{endOf}(W_{i,j,P_{ij}}) \right)$$

$$\text{interval } W_{ij\ell}, \text{ size} = 1 \qquad \forall i, \forall j, \forall \ell \in \{1, \ , P_{ij}\} \quad \text{(V1)}$$

$$\text{interval } O_{kc\lambda}, \text{ optional }, \text{size} = 1 \qquad \forall k, \forall c \in \text{MaxW}_k, \forall \lambda \in \{1, \ , \delta_c^{\text{lo}}\} \quad \text{(V2)}$$

$$O_{kc\lambda} \subseteq [u_c, v_c] \qquad \forall k, \forall c \in \text{MaxW}_k, \forall \lambda \in \{1, \ , \delta_c^{\text{lo}}\} \quad \text{(C1)}$$

$$\text{EndBeforeStart}(W_{ij,\ell-1}, W_{ij,\ell}) \qquad \forall i, \forall j, \forall \ell \in \{2, \ , P_{ij}\} \quad \text{(C2)}$$

$$\text{EndBeforeStart}\left(W_{i,j,p_{ij}}, W_{i,j+1,0}\right) \qquad \forall i, \forall j \in \{0, \ , n_i - 1\} \quad \text{(C3)}$$

$$\text{NoOverlap}\left(\underset{\forall i,j \in \text{T}_k,\ell}{[W_{ij\ell}]} \ \cup \ \underset{\forall c \in MaxW_k,\lambda}{[O_{kc\lambda}]} \right) \qquad \forall k \quad \text{(C4)}$$

$$v_c - u_c - \sum_{\lambda=1,\ldots,\delta_c^{\text{lo}}} \left(O_{kc\lambda} \in [u_c, v_c]\right) \leq \delta_c \qquad \forall k, \forall c \in \text{MaxW}_k \quad \text{(C5)}$$

3 Experiments

We evaluate our models on 78 pJSP instances (Juvin *et al.* 2023), each enriched with 9 sets of MaxW constraints, yielding 936 instances. All instances and generation explanations are available on GitHub. All experiments are conducted with a time limit of 45 minutes per instance. Results clearly demonstrate the effectiveness of our Mistral-based approach. Over the full benchmark set, Mistral solves 171 instances to optimality (24.3%), compared to only 35 instances (4.98%) for CPO. Furthermore, as illustrated in Figure 2, the Mistral

model consistently produces better solutions. In particular, it achieves solutions with a gap below 20% on 565 instances (80.4%) to the best solution found, whereas CPO reaches this level of solution quality on only 233 instances (33.2%). Overall, these results highlight the strong performance of the proposed CP model for the MaxW-pJSP. By directly integrating workload constraints within a preemptive scheduling framework and leveraging specialized propagation mechanisms, our approach significantly outperforms a state-of-the-art industrial solver on a large and challenging benchmark set.

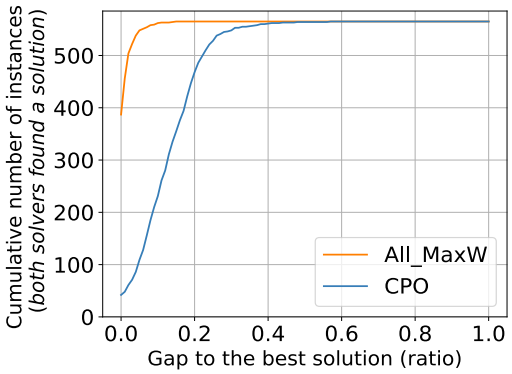


Fig. 2: Cumulative distribution of the gap to the best solution found by any solver

4 Conclusion and future work

In this paper, we have validated an efficient CP-based approach for handling MaxW constraints, including scenarios with a large number of constraints, which are common in production due to rolling rules (e.g., "no more than 5 working days over any 7-day window"). Our experiments demonstrate that the proposed Mistral model scales well and significantly outperforms a state-of-the-art solver (CPO) on an extensive benchmark set. Future work includes exploring a MILP-based formulation to analyze modeling and performance trade-offs. **Acknowledgements** This work was supported by the French National Research Agency (ANR) under the project **HIS**^{3} ANR-22-CE10-0012-05.

References

Pinedo, M.L., 2012, "Scheduling: Theory, Algorithms, and Systems", *Springer*, New York.

Laborie, P., Rogerie, J., Shaw, P., Vilím, P., 2018, "IBM ILOG CP Optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG", *Constraints*.

Strandmark, P., Qu, Y., Curtois, T., 2020, "First-order linear programming in a column generation-based heuristic approach to the nurse rostering problem", *Computers & Operations Research*.

Liu, Z., Liu, Z., Zhu, Z., Shen, Y., Dong, J., 2018, "Simulated annealing for a multi-level nurse rostering problem in hemodialysis service", *Applied Soft Computing*.

Mauguière, Ph., Billaut, J.-C., Bouquard, J.-L., 2005, "New single machine and job-shop scheduling problems with availability constraints", *Journal of Scheduling*.

Müller, D., Kress, D., 2022, "Filter-and-fan approaches for scheduling flexible job shops under workforce constraints", *International Journal of Production Research*.

Lunardi, W.T., Birgin, E.G., Laborie, P., Ronconi, D.P., Voos, H., 2020, "MILP and CP models for the online printing shop scheduling problem", *Computers & Operations Research*.

Juvin, C., Hebrard, E., Houssin, L., Lopez, P., 2023, "An efficient constraint programming approach to preemptive job shop scheduling", *Proceedings of CP 2023*.

Interpretability of optimization solutions in preventive maintenance rescheduling

Liwen Zhang[1], Kévin Ducharlet[1], Sara Maqrot[1], Houssem Saidi[1]

Berger-Levrault
{first name.last name}@berger-levrault.com

Keywords: interpretability, rescheduling, maintenance project management

1 Introduction

The industry 5.0 revolution is founded on a fundamental transformation in the interactions between humans and intelligent technologies. Maintenance project management fits naturally within this context, since it plays a crucial role in preventing machine breakdowns that disrupt supply chains and reduce customer satisfaction.

Integrating human-centric principles into the maintenance scheduling process is important when static schedules require real-time adjustments, which often happens during operational days. In such cases, decision-makers need clear insights about the quality of the revised schedule, including why it may be better or worse than the original one. In this work, we address a maintenance scheduling problem formulated as a Constraint Satisfaction Problem (CSP) and introduce a framework that places the decision-maker at the center of the process. The goal is to support decision-makers in selecting appropriate strategies when the initial schedule needs to be modified due to unexpected events. The framework compares the original and revised schedules by highlighting the differences in global scores and detailing how each solution violates the relevant constraints.

2 Problem description

We focus on Preventive Maintenance (PM), which is a key activity in maintenance project management. In PM scheduling and rescheduling, the goal is to plan and organize PM tasks in an efficient manner to prevent unexpected equipment failures and keep high operational performance in manufacturing systems. Addressing this class of problems requires the decision-maker to answer two questions: 1) When should each maintenance task start? 2) Which skilled technician should perform the PM task?

Given the planning horizon (daily, weekly or monthly) with p days (z weeks when the horizon exceeds one week), q available staff members and r PM tasks to perform defined by the decision-makers of the organization, together with the daily operating hours $\mathcal{H} = [\tau, \upsilon]$, each staff member $s \in S$ is characterized by an unavailability interval $[n_s, m_s]$, a daily workload limit λ_s, and a weekly workload limit ω_s to respect. Each staff member also has a specialization α_s, which determines the type of PM tasks they are qualified to perform. On the other side, a set of PM tasks T must be executed within the defined planning

horizon. For each task $t \in T$, a deadline γ_t indicating the allowable completion time, and a duration δ_t are specified. In addition, each task requires a specific specialization α_t^*, which must be matched by the specialization of the staff member assigned to execute it.

The problem contains two types of variables. The first variable, $start_t$, represents the start time of each PM task t, with a domain defined as $start_t \in \left[0,\ \frac{24\times3600\times p}{g}\right]$, where g denotes the planning granularity (in seconds), used to convert all time-related parameters into an integer time-slot representation. The second variable, ass_t, denotes the staff member assigned to task t, with a domain $ass_t \in \{s_1, s_2, ..., s_q\}$. The problem is subject to the following five constraints, each associated with a penalty applied to either the hard score or the soft score, depending on the nature of the constraint:

C_1: Respect the daily opening hours of the organization.

$$\left(start_t < \tau \ \vee \ start_t > \upsilon\right) \ \Rightarrow \ score_{hard} \vee score_{soft} = -1. \tag{1}$$

C_2: Respect the unavailability periods of staff members.

$$\left(start_t \in [\, n_s, m_s \,]\right) \ \Rightarrow \ score_{hard} \vee score_{soft} = -1. \tag{2}$$

C_3: Ensure that each task is assigned to a staff member with the required specialization.

$$\left(\alpha_{\,ass_t} \neq \alpha_t^*\right) \ \Rightarrow \ score_{hard} \vee score_{soft} = -1. \tag{3}$$

C_4: Respect the daily and weekly workload limits of each staff member.

$$\left(\sum_{t:\, start_t = d,\ ass_t = s} \delta_t > \lambda_s\right) \ \Rightarrow \ score_{hard} \vee score_{soft} = -1. \tag{4}$$

$$\left(\sum_{t:\, ass_t = s} \delta_t > \omega_s\right) \ \Rightarrow \ score_{hard} \vee score_{soft} = -1. \tag{5}$$

C_5: Ensure that each PM task must be scheduled and completed before its assigned deadline.

$$\left(start_t + \delta_t > \gamma_t\right) \ \Rightarrow \ score_{hard} \vee score_{soft} = -1 \tag{6}$$

3 Rescheduling with solution interpretability

When a schedule is generated according to the model presented previously, it may still be disrupted by unexpected events that affect the initial schedule (Details of the applied rescheduling procedures are provided in our previous work [Ducharlet et al., 2025]). In our study, we classify these events into two groups.

1. *Events that lead to modifications of the problem.* In this group, we consider three types of situations:

  - *Unexpected staff unavailability or availability.* A staff member may take unexpected leave due to personal reasons, or an additional staff member may become available on the operational day. Such events change the set of available staff when the schedule is regenerated.
  - *Addition or removal of PM tasks on the operational day.* This alters the set of PM tasks that must be scheduled when generating a new schedule.
  - *Changes in constraint configurations or parameter values.* This includes activating or deactivating some constraints, modifying the nature of a constraint (e.g., switching one activated constraint from hard to soft), or adjusting constraint-related parameters (e.g., changing a weekly workload limit ω_s from 35h to 37h).
2. *Events that lead to modifications of the solution.* In this group, we consider one situation:
  - *Adjustment of the schedule based on decision-makers' expertise.* Decision-makers are domain experts. They may adjust the schedule generated by the optimizer without providing a formal justification. Such adjustments are often based on practical experience, such as deciding that a specific PM task should be performed by a particular staff member.

To address the two identified groups of uncertain events, we propose two rescheduling strategies:

- *Full recovery.* The optimizer is rerun to generate a completely new schedule after adjustments due to unexpected events, potentially involving changes in the set of PM tasks, staff availability, or constraint configurations. This strategy is only applicable to group 1 events. First Fit is used for initialization, while Simulated Annealing is employed for local-search-based optimization.
- *Dynamic scheduling.* The optimizer is rerun to generate a revised schedule while pining certain assignments from the initial schedule. This partial rescheduling approach is particularly appropriate for group 2 events, but can also be applied to group 1 events when specific task assignments must be preserved. No initialization is performed in this strategy, Simulated Annealing is employed for local-search-based optimization.

To support the interpretability of changes between two solutions, we develop a framework with a Human-Machine Interface (HMI) that highlights differences between the initial and adjusted schedules. The HMI first shows changes in the global score (e.g., 0hard/–2soft), allowing users quickly see if the overall quality has decreased. A second view provides a detailed comparison of constraint violations, including the number of violations and the change in violation scores. Negative impacts are shown in red, and improvements of score are in green. By reviewing these elements, decision-makers can decide whether to accept the adjusted schedule or generate a new one using the proposed rescheduling strategies. An illustration of this HMI is provided in the next section through a case study.

4 Interpretability module in practice: a case study

We conduct a case study in this section to demonstrate the interpretability module of our framework. The problem parameters are as follows: $p = 2$, $q = 5$,

$r = 30$ $\lambda_s = 7h$, $[\tau, \upsilon] = [7h, 17h]$, $\alpha_s, \alpha_t^* \in \{sp\_a, sp\_b\}$, $\delta_t \in \{1h, 1.5h, 2h\}$. The parameters $[n_s, m_s]$ and γ_t (applicable to 50% of the PM tasks and set no earlier than the beginning of the second day) are presented directly within the framework interface. The activated constraints in this example are $C_1 = \{hard\}$, $C_2 = \{hard\}$, $C_3 = \{soft\}$, $C_4 = \{hard\}$, $C_5 = \{soft\}$.

As shown in Part 1 of Figure 1, we first generate the initial schedule using the optimizer introduced in our previous work [Ducharlet et al., 2025]. The right side of the figure displays the detailed score of this solution with no constraint violations. We then introduce a one-day unavailability on the second day for the fourth staff member (part 2 of Figure 1), which reduces the global score from 0hard/0soft to –3hard/0soft, meaning the schedule is no longer feasible. To resolve this issue, we pin all other assigned PM tasks and reschedule only the three tasks causing violations. Part 3 of Figure 1 presents the repaired schedule, with the score improving from –3 to 0 on the hard component and a new soft violation (C_3: Respect_specialization: –2). However, the solution becomes feasible after applying the dynamic scheduling strategy.

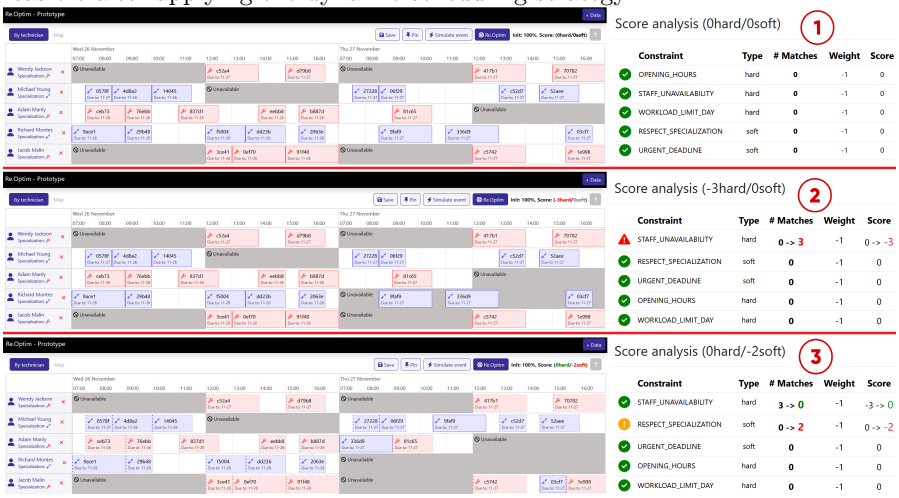


Fig. 1. Illustration of the interpretability module based on a case study.

5 Conclusion

In this work, we present the maintenance scheduling problem and a rescheduling framework that handles unexpected events. The framework includes interpretability in a HMI that helps decision-makers understand the adjusted schedule through score comparisons and detailed constraint violations. In future work, we aim to use agentic AI to transform these interpretations into natural language explanations that are easier for end users to understand.

References

[Ducharlet et al., 2025] Ducharlet, K., Zhang, L., Maqrot, S., and Saidi, H. (2025). A recommendation system-based framework for enhancing human-machine collaboration in industrial timetabling rescheduling: Application in preventive maintenance. In *Working Conference on Virtual Enterprises*, pages 363–381. Springer.

A three-phases matheuristic for the SALB3PM problem

Thiago Giachetto de Araujo, Matthieu Py, Laurent Deroussi and Nathalie Grangeon

University of Clermont Auvergne, CNRS, Clermont Auvergne INP,
Mines Saint-Etienne, LIMOS, 63000 Clermont-Ferrand, France
{thiago.giachetto_de_araujo,matthieu.py,laurent.deroussi,nathalie.grangeon}@uca.fr

Keywords: Scheduling, Line Balancing, SALBP, Power Peak Minimization, Matheuristics, Energy, Constrained Programming.

1 Introduction

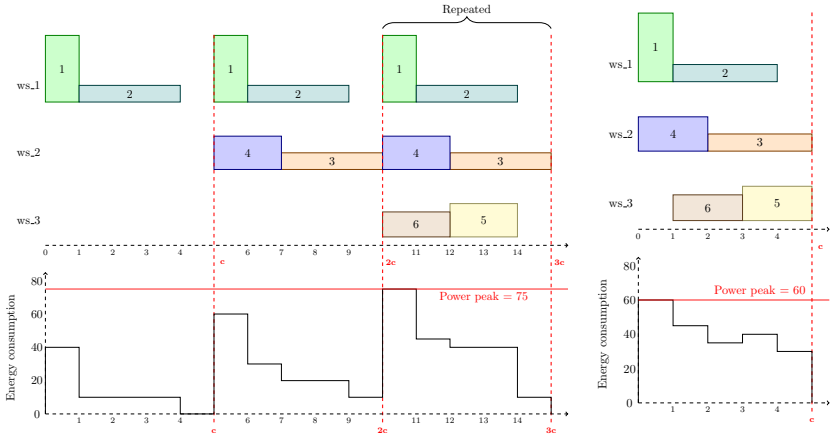
Assembly lines represent fundamental production systems characterized by sequentially arranged workstations. In these systems, unfinished products flow linearly through workstations, with specific tasks being executed at each workstation until the product is completed. Such systems are commonly modeled as *Simple Assembly Line Balancing Problem* (SALBP) (Baybars 1986), which has been extensively studied in the literature (Scholl and Voss 1997, Fathi *et al.* 2018).

The growing urgency to address climate change has emphasized the need to reduce industrial energy consumption. According to the International Energy Agency (IEA), industrial operations account for 45% of global CO_2 emissions and 39% of final energy consumption (IEA 2024). This environmental imperative has motivated the development of energy-aware extensions to the classical SALBP, incorporating sustainability considerations into production line optimization.

An example is SALBP with Power Peak Minimization (SALB3PM) introduced by (Gianessi *et al.* 2019). This variant associates each task with a fixed power consumption and aims to minimize power peak consumption throughout the production cycle.

2 The SALBP with Power Peak Minimization

In the *Simple Assembly Line Balancing Problem with Power Peak Minimization*, a set $\mathcal{O}$ of n *tasks* should be assigned to a set $\mathcal{M}$ of m *workstations* arranged in sequence.



(a) Three first cycle times of operation

(b) After stabilization

Fig. 1: Example of solution for the SALB3PM

Indivisible operations, called tasks, of the set $\mathcal{O}$ are necessary to assemble a product. Each task should be assigned to a unique workstation. For each workstation, c units of time, known as *cycle time*, are available. Each task $j \in \mathcal{O}$ is associated with a known processing time t_j and power consumption w_j. Processing times and power consumption are constant and independent of the assigned workstation. Given production constraints, tasks must observe precedence requirements. Tasks assigned to a workstation cannot overlap. Tasks must finish by the cycle time c. In the SALB3PM, given a known cycle time and the maximum number of workstations, the objective is to assign and schedule all tasks, aiming to minimize the peak of power consumption (Gianessi *et al.* 2019). In the SALB3PM, inserting idle time before tasks (*delay*) could be necessary to reduce the power peak.

Figure 1a shows the first three cycles of a production line. In the first cycle, assembly of the first product begins. In the second cycle, the assembly of the first product continues, and the production of a new product starts. In the third cycle, we are manufacturing three products. At the end of this cycle, the first product is finished. We observe that the energy consumption of an instant is the summation of the energy consumption of each task executed at that instant. Figure 1b shows a solution with an idle time before task 6. We can observe a reduction in the power peak.

3 Proposed method

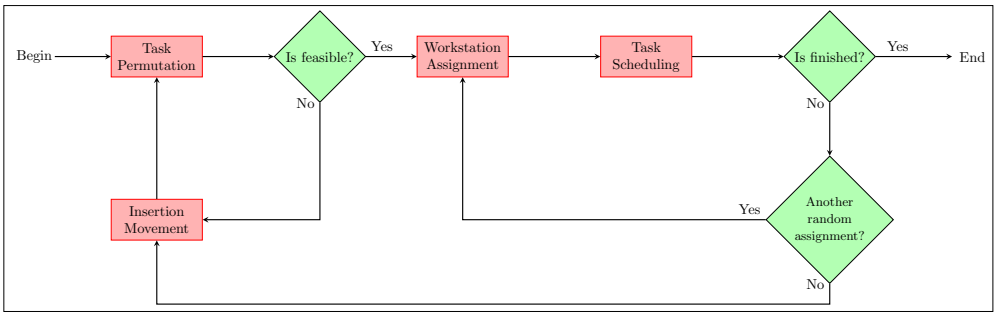


Fig. 2: Diagram of a three-phases solver for the SALB3PM.

The SALB3PM is NP-hard; therefore, we propose to divide the problem into three subproblems: task sequencing, workstation assignment, and task scheduling. As shown by Figure 2, we propose a matheuristic scheme with three components. Insertion movement is used as a metaheuristic component for task sequencing, dynamic programming builds workstation assignments, and task scheduling is solved with mathematical programming.

3.1 Permutation phase

The first phase considers only task permutations $\sigma = (\sigma_1, \ldots, \sigma_n)$ that follow the precedence constraints. Initially, a random permutation that satisfies the precedence constraints is generated. After, a neighbor permutation is built by an insertion movement that respects the precedence constraints. At the end of this phase, the feasibility problem SALBP-F, with m workstations and cycle time c, is solved with a greedy algorithm issued from (Scholl and Voss 1997). A permutation proceeds to the second phase if a feasible assignment regarding the cycle time exists.

3.2 Assignment phase

In the second phase, a permutation σ is received. Feasible assignments that respect the permutation are randomly generated. To accomplish this, we adapt the idea from the giant tour splitting problem from the vehicle routing problem (Prins 2004) and sequence splitting from the SALBP (Lahrichi 2022). We can transform it into the equivalent problem of finding a path in a graph with at most m arcs. This graph contains $n+1$ nodes. The first node is a dummy node, and the others represent the tasks. An arc (i, j) in this graph exists if assigning the tasks $\sigma_{i+1}, \ldots, \sigma_j$ to the same workstation follows the cycle time constraint. The problem of generating assignments with at most m workstations is equivalent to finding a path with at most m arcs, from node 0 to node n.

The computation of possible paths is realized by adapting a multi-label extension of Bellman's algorithm (Desrochers 1988). A label λ for a node j contains the length in number of arcs of the path to arrive at node j. For each label, the number of paths $\mathcal{C}_j(\lambda)$ and the set of predecessors $\mathcal{P}_j(\lambda)$ are stored. As the objective is to generate all feasible paths, just labels with less than m arcs are stored on nodes $1, \ldots, n-1$. $\mathcal{C}_j(\lambda)$ and $\mathcal{P}_j(\lambda)$ are updated with each new path with k arcs to reach at node j is found. A random assignment is built from the last node. Until the node 0 is reached, each predecessor l is chosen with probability proportional to the number of paths to achieve the predecessor.

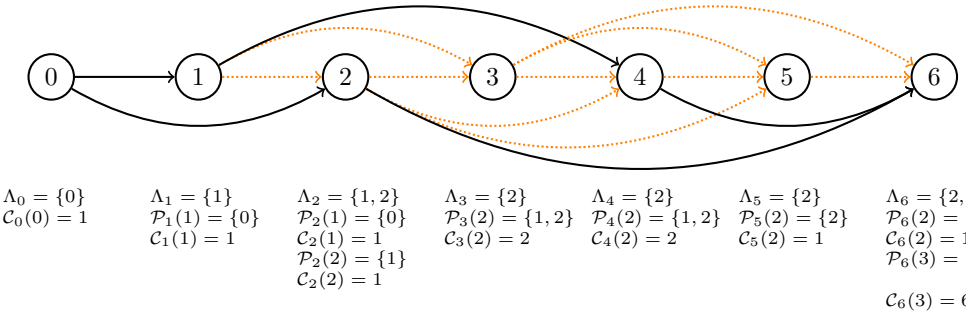


Fig. 3: **Examples of feasible paths**

Figure 3 shows all the possible paths in the graph. For each node $\mathcal{C}_j(\lambda)$ and $\mathcal{P}_j(\lambda)$ is shown. Two paths are highlighted: one with two and the other with three arcs. The path with three arcs $0-1-4-6$ is equivalent to the assignment $\{\{\sigma_1\}, \{\sigma_2, \sigma_3, \sigma_4\}, \{\sigma_5, \sigma_6\}\}$. And the path $0-2-6$ is equivalent to $\{\{\sigma_1, \sigma_2\}, \{\sigma_3, \sigma_4, \sigma_5, \sigma_6\}\}$.

3.3 Task scheduling

The sequence of tasks is built in the first phase, and the assignment is chosen in the second. In this phase, the starting time of each task remains to be defined. Here, we present a constrained optimization formulation to address this problem. Table 1 gives the parameters and decision variables used.

Constraints 2 ensure that tasks maintain the order of execution. Constraints 3 guarantee that the last task of the workstation finishes before the cycle time. Constraint 4 and the objective function 1 ensure the variable W contains the peak of energy consumption. For each time frame t, the power peak W is greater than or equal to the summation of the power consumption w_j of tasks being executed at t.

Table 1: Parameters and decision variables.

Parameters	
σ_j^k	j-th task assigned to workstation k
$\mathcal{O}_k$	permutation $\{\sigma_1^k, \sigma_2^k, \cdots, \sigma_{n_k}^k\}$ assigned to workstation $k \in \mathcal{M}$
n_k	is the number of tasks, $\lvert\mathcal{O}_k\rvert$, assigned to a workstation $k \in \mathcal{M}$
$\gamma_{min}^{\sigma_i^k}$	$\sum_{j=1}^{i-1} t_{\sigma_j^k}, \forall i \in \{1, \cdots, n_k\}, k \in \mathcal{M}$ is the earliest starting time of task σ_i^k
$\gamma_{max}^{\sigma_i^k}$	$c - \sum_{j=i}^{n_k} t_{\sigma_j^k}, \forall i \in \{1, \cdots, n_k\}, k \in \mathcal{M}$ is the latest starting
Decision variables	
S_j	is the starting time of task j
W_{max}	is the maximum power consumed in all period

$$\min W \tag{1}$$

s.t.

$$S_{\sigma_i^k} + t_{\sigma_i^k} \leq S_{\sigma_{i+1}^k}, \qquad \forall k \in \mathcal{M}, i \in [1, n_k - 1] \tag{2}$$

$$S_{\sigma_{n_k}} + t_{\sigma_{n_k}} \leq c, \qquad \forall k \in \mathcal{M} \tag{3}$$

$$\sum_{j \in \mathcal{O}} ((S_j \geq \tilde{t} - t_j + 1) \wedge (S_j \leq \tilde{t}))\, w_j \leq W, \qquad \forall \tilde{t} \in [0, c-1] \tag{4}$$

$$S_{\sigma_i^k} \in [\gamma_{min}^{\sigma_i^k}, \gamma_{max}^{\sigma_i^k}], \qquad \forall k \in \mathcal{M}, i \in \mathcal{O}_k \tag{5}$$

$$W \geq 0. \tag{6}$$

4 Computational experiments

We evaluate our approach in a benchmark of the literature (Gianessi *et al.* 2019, Py *et al.* 2024) with 25 instances, 19 of which have known optimal solutions. For each configuration and instance, 10 independent runs were executed. We compare Integer Linear Programming (Gianessi *et al.* 2019) and Constraint Programming formulations for the delay phase. Considering instances with known optimal solutions, our best configuration achieved 15 optima, with an average optimality gap of less than 0.51

Acknowledgements

This study was supported in part by the International Research Center "Innovative Transportation and Production Systems".

References

Baybars I., 1986, "A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem", *Management Science*, Vol. 32, pp. 909–932.

Desrochers M., 1988, "An algorithm for the shortest path problem with resource constraints", *GERAD, Ecole des HEC*, Canada, G-88-27.

Fathi M., D.B.M.M. Fontes and M. Urenda Moris and M. Ghobakhloo, 2018, "Assembly line balancing problem: A comparative evaluation of heuristics and a computational assessment of objectives", *Journal of Modelling in Management*, Vol. 13, pp. 455-474.

Gianessi P., X. Delorme and O. Masmoudi, 2019, "Simple Assembly Line Balancing Problem with Power Peak Minimization", in Ameri F. *et al. Advances in Production Management Systems. Production Management for the Factory of the Future*, Vol. 566, pp. 239-247.

International Energy Association, 2024, "World Energy Outlook 2024", *International Energy Association.*

Lahrichi Y., 2022, "Balancing reconfigurable or robotic assembly lines: exact and hybrid methods", Ph. D. Thesis *University of Clermont Auvergne.*

Prins C., 2004, "A simple and effective evolutionary algorithm for the vehicle routing problem", *Computers & Operations Research*, Vol. 31, pp. 1985-2002.

Py M., A. Tuyaba, L. Deroussi, N. Grangeon and S. Norre, 2024, "Application of SAT to the Simple Assembly Line Balancing Problem with Power Peak Minimization", *15th Pragmatics of SAT international workshop, a workshop of the 27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024).*

Scholl A., S. Voss, 1997, "Simple assembly line balancing - Heuristic approaches", *Journal of Heuristics*, Vol. 02, pp. 217–244.

I - Scheduling under uncertainty

Stability in Stochastic Project Scheduling: Comparing Variable and Fixed Resource Allocation

Maria Elena Bruni[1] and Öncü Hazır[2]

[1] Department of Mechanical Energy and Management Engineering, Rende, Italy
mariaelena.bruni@unical.it
[2] Rennes School of Business, Rennes, France
oncu.hazir@rennes-sb.com

Keywords: Project Management, Project Scheduling, Stochastic programming, Optimization, Schedule Robustness.

1 Introduction

This research develops a comprehensive optimization framework for generating stable project schedules under activity duration uncertainty. Schedule stability is measured by deviations between the planned and actual schedule. We investigate the stability regarding two perspectives: the changelessness of the activity start times and the resource allocation. Stability of both types is desired in practice to limit the system nervousness and additional work and cost. We formulate two-stage stochastic programming models that directly optimize schedule stability measures rather than surrogate slack-based metrics common in the literature. We present two model variants representing different organizational contexts and constraints. The Fixed Resource Allocation (FRA) model suits projects where resource reallocation is prohibitively expensive (e.g., moving heavy equipment between geographically distant construction sites), while the Variable Resource Allocation (VRA) model accommodates agile environments where resource flexibility provides value (e.g., software development with shared teams). We explicitly model and quantify both temporal stability ($\mathscr{M}_1$: expected weighted deviations in activity start times) and resource stability ($\mathscr{M}_2$: expected changes in resource flow allocations). Through extensive computational experiments, we provide actionable insights on how problem characteristics (deadline tightness, network complexity, resource factor, resource strength) affect the tradeoff between temporal and resource stability. We compare our approach against the established baseline of minimizing extra precedence arcs Deblaere et al. (2007), through a rigorous out-of-sample validation with 1,000 scenarios, demonstrating substantial improvements in both stability measures.

2 Mathematical models

This section presents our two-stage stochastic programming framework for generating stable project schedules. We begin with the baseline resource-constrained project scheduling problem (RCPSP), then we systematically extend it to incorporate uncertainty and stability measures.

Consider a project network described by an activity-on-node network, where the set $V = \{1, \ldots, n\}$ represents a set of activities to be scheduled, and the dummy activities $s = 0$ and $t = n + 1$ represent, respectively, the project starting and ending nodes. The set E denotes the finish-start, zero-lag precedence relationships among activities. Specifically, $(i, j) \in E$ requires that activity i be a predecessor activity of activity j. The execution of project activities requires renewable resources. The resource availability of $k \in K$ every time period is R_k. Each activity $i \in V$ requires a non-negative amount r_{ik} of each resource $k \in R$. On this basis, and assuming that each activity $i \in V$ has a deterministic duration d_i, and $d_s = d_t = 0$, the compact flow-based continuous-time formulation of Artigues et al. (2003) can be defined for the RCPSP. In this model, flow variables f_{ijk} are introduced to model the flow of resources for each type of resource and

for each pair of nodes and denote the number of units of resources k directly transferred from an activity i to an activity j. These variables eliminate the forbidden-set constraints, provided that $r_{sk} = r_{tk} = R_k \ \forall k \in K$. The binary variables x_{ij} assume a value equal to one if an extra arc is introduced into the flow network, implying that activity j is restricted to start after the completion of activity i. Variables $S_i, i = 0, \ldots n+1$ represent the starting times of activities.
Under these assumptions, the formulation for the RCPSP can be written as follows:

$$\min S_{n+1} \tag{1}$$
$$x_{ij} = 1 \ \forall (i,j) \in E \tag{2}$$
$$S_j - S_i \geq d_i - M(1 - x_{ij}) \ \forall i, j \in V, i \neq j \tag{3}$$
$$S_0 = 0 \tag{4}$$
$$f_{ijk} - N x_{ij} \leq 0 \ \ \forall i, j \in V, i \neq j, i \neq n+1, j \neq 0, \ \forall k \in K, \tag{5}$$
$$\sum_{j \in V \setminus \{i\}} f_{ijk} = r_{ik} \ i \in V \setminus \{n+1\}, \ \forall k \in K, \tag{6}$$
$$\sum_{i \in V \setminus \{j\}} f_{ijk} = r_{jk} \ j \in V \setminus \{0\}, \ \forall k \in K, \tag{7}$$
$$f_{ijk} \geq 0 \ \forall i, j \in V, i \neq j, i \neq n+1, j \neq 0, \forall k \in K \tag{8}$$
$$x_{ij} \in \{0,1\} \ \forall i, j \in V, i \neq j \tag{9}$$

The preprocessing constraints (3) set the preexisting precedence constraints. The constraints (3) and (4) impose precedence constraints and set the project's start time, respectively. M is an arbitrarily large number. Constraints (5) are big-M constraints linking flow variables with precedence-related variables. Here N can be set to $min(r_{ik}, r_{jk})$. Constraints (6) and (7) are the resource flow-conservation constraints. The remaining constraints define the nature of the variables.

In practice, activity durations are uncertain. Let Σ denote a finite set of scenarios representing possible realizations of activity durations. Scenario $\sigma \in \Sigma$ has probability π_σ (with $\sum_{\sigma \in \Sigma} \pi_\sigma = 1$) and specifies duration d_i^σ for each activity i. In FRA, the second-stage recourse is limited to adjusting activity start times; resource flows and precedence relationships remain fixed at their first-stage values. The model reads as follows:

$$(FRA) \ \ \min \sum_{j \in V} \sum_{\sigma \in \Sigma} \pi_\sigma (S_j^{\sigma+} + S_j^{\sigma-})$$
$$(3) - (9)$$
$$(S_j + \Delta_j^\sigma) - (S_i + \Delta_i^\sigma) \geq d_i^\sigma - M[1 - x_{ij}] \ \forall i, j \in V, i \neq j, \forall \sigma \in \Sigma \tag{10}$$
$$S_{n+1} \leq D \tag{11}$$
$$\Delta_i^\sigma = S_i^{\sigma+} - S_i^{\sigma-} \ \forall i \in V \setminus \{0\}, \forall \sigma \in \Sigma \tag{12}$$
$$S_i^{\sigma-} \geq 0, S_i^{\sigma+} \geq 0, \forall i, j \in V, \forall \sigma \in \Sigma \tag{13}$$

The objective function minimizes the weighted sum of the expected absolute deviations between the start times in the realized schedule and those in the baseline schedule ($\mathscr{M}_1$). Constraint (11) bounds the makespan by deadline D, generally stipulated by the client or contractor. Constraints (10) adjust the starting times of the successor activities j, according to the realization of the duration of activity i in scenario σ. Constraints (12) define the variable Δ_i^σ, introduced for conciseness. The railroad reactive scheduling, requiring $S_j^{Realized} \geq S_j^{Planned}$ can easily be incorporated into the model by removing the variables $S_j^{\sigma-}, \forall j \in V, \forall \sigma \in \Sigma$. The Anchor-Robust RCPSP was defined in Bendotti et. al. (2023), within the general framework of 2- stage robustness (or adjustable robustness). In this framework, a baseline solution is chosen in first stage, then, the schedule can be changed in second stage into a new schedule with sequencing and resource allocation decisions unchanged. The starting times of anchored jobs are the same in baseline schedule and second-stage schedule. Our model is a more general approach than the anchor robustness approach, and it is not a robust approach.

To present the mathematical model of the VRA version, let define ${x_{ij}}^{\sigma+}$ equal to one if a new precedence relation has been established between i and j in the second stage that was not present in the first stage and ${x_{ij}}^{\sigma-}$ if a precedence relation between i and j, present in the first stage has been removed in the second stage. Thus, the two-stage stochastic programming model is presented as the following model.

$$(VRA) \quad \min \sum_{j \in V} \sum_{\sigma \in \Sigma} \pi_\sigma (S_j^{\sigma+} + S_j^{\sigma-}) \tag{14}$$

$$(3) - (9), (11) - (13)$$

$$(S_j + \Delta_j^\sigma) - (S_i + \Delta_i^\sigma) \geq d_i^\sigma - M[1 - (x_{ij} + {\xi_{ij}}^\sigma)]$$

$$\forall i, j \in V, i \neq j \tag{15}$$

$$f_{ijk} + \phi_{ijk}^\sigma \leq N(x_{ij} + {\xi_{ij}}^\sigma) \forall i, j \in V, i \neq j, i \neq n+1, j \neq 0$$

$$\forall k \in K, \ \forall \sigma \in \Sigma \tag{16}$$

$$\sum_{j \in V \setminus \{i\}} (f_{ijk} + \phi_{ijk}^\sigma) = r_{ik} \ i \in V \setminus \{n+1\}, \forall k \in K \tag{17}$$

$$\sum_{i \in V \setminus \{j\}} (f_{jik} + \phi_{ijk}^\sigma) = r_{jk} \ j \in V \setminus \{0\}, \forall k \in K \tag{18}$$

$${x_{ij}}^{\sigma+} + x_{ij} \leq 1 \ \forall i, j \in V, i \neq j, \forall \sigma \in \Sigma \tag{19}$$

$${x_{ij}}^{\sigma-} \leq x_{ij} \ \forall i, j \in V, i \neq j, \forall \sigma \in \Sigma \tag{20}$$

$$\xi_{ij}^\sigma = {x_{ij}}^{\sigma+} - {x_{ij}}^{\sigma-} \ \forall i, j \in V, i \neq j, \tag{21}$$

$$\phi_{ijk}^\sigma = f_{ijk}^{\sigma+} - f_{ijk}^{\sigma-} \forall i, j \in V, i \neq j, i \neq n+1, j \neq 0, \forall k \in K, \forall \sigma \in \Sigma \tag{22}$$

$${f_{ijk}}^{\sigma+}, {f_{ijk}}^{\sigma-} \geq 0 \forall i, j \in V, i \neq j, i \neq n+1, j \neq 0, \forall k \in K, \forall \sigma \in \Sigma \tag{23}$$

$${x_{ij}}^{\sigma+}, {x_{ij}}^{\sigma-} \in \{0, 1\} \ \forall i, j \in V, i \neq j, \forall \sigma \in \Sigma \tag{24}$$

Constraints (3)-(9) are first-stage constraints and only involve first-stage variables. Constraints (15) are the equivalent of constraint (10), but consider the possibility that different extra arcs can be used for different disruption scenarios. Constraints (16)-(18) ensure resource feasibility in every disruption scenario. It should be mentioned that this condition enforces the complete recourse property, i.e., the second stage problem is feasible for all choices of the first stage variables. Constraints (19) and (20) are logical constraints stating that an Extra arc can be added in the second stage only if it was not already added in the first stage, whilst it can be deleted only if it was present in the first stage. Constraints (21)-(22) properly define the variables ξ_{ij}^σ and ϕ_{ijk}^σ introduced for the sake of conciseness.

2.1 Model Comparison

Given our variable definition $\mathscr{M}_1 = \sum_{j \in V} \sum_{\sigma \in \Sigma} \pi_\sigma (S_j^{\sigma+} + S_j^{\sigma-})$ and $\mathscr{M}_2 = \sum_{\sigma \in \Sigma} \pi_\sigma \sum_{i,j \in V, k \in K} (f_{ijk}^{\sigma+} + f_{ijk}^{\sigma-})$. The lower are the values of the measures, the higher is the stability. In our experiments, each instance's project due date D is set as Cmax $(1 + \alpha)$. Cmax represents the optimal makespan under a deterministic environment. Table 1 only reports the results averaged over different network characteristics, for the sake of conciness.

Table 1: Average stability values for different values of α.

	VRA		FRA		Extra	
	$\mathscr{M}_1$	$\mathscr{M}_2$	$\mathscr{M}_1$	$\mathscr{M}_2$	$\mathscr{M}_1$	$\mathscr{M}_2$
$\alpha = 0.1$	0.90	10.36	0.97	11.44	4.49	13.48
$\alpha = 0.2$	0.43	13.19	0.81	10.75	3.34	10.18
$\alpha = 0.3$	0.22	11.69	0.26	8.14	3.82	6.86

The VRA model consistently achieves superior temporal stability across all deadline tightness levels, with $\mathscr{M}_1$ values of 0.90, 0.43, and 0.22 for $\alpha \in \{0.1, 0.2, 0.3\}$ respectively. Comparing VRA to the Extra benchmark (which simply minimizes additional precedence arcs without explicit stability optimization), the improvements are dramatic: 80% improvement for $\alpha = 0.1$, 87% improvement for $\alpha = 0.2$ and 94% improvement for $\alpha = 0.3$. These substantial gains validate our core hypothesis: explicitly optimizing stability measures during baseline generation, rather than using surrogate heuristics, yields dramatically more stable schedules. Even compared to FRA (which also optimizes stability but with resource constraints), VRA achieves 7-52% better temporal stability depending on α. However, this temporal stability improvement comes at a cost in resource stability, and there is a trade-off between the two types of stability. The tradeoff becomes evident as deadlines relax: When $\alpha = 0.2$ $\mathscr{M}_2 = 13.19$ for VRA and $\mathscr{M}_2 = 10.75$ for FRA (23% worse resource stability). When $\alpha = 0.3$ $\mathscr{M}_2 = 11.69$ for VRA and $\mathscr{M}_2 = 8.14$ for FRA (44% worse resource stability). Meanwhile, comparing temporal stability at $\alpha = 0.3$, VRA achieves $\mathscr{M}_1 = 0.22$ while FRA 0.26-only 15% better. While for $\alpha = 0.2$ the starting time stability drop (increase of $\mathscr{M}_1$ of around 84%) is not compensated by a notable resource allocation stability increase (the $\mathscr{M}_2$ is only 20% lower), when $\alpha = 0.3$, the FRA model can be considered a good option for the decision maker. At this relaxed deadline, FRA becomes highly competitive, achieving comparable temporal stability with substantially superior resource stability. For tight deadlines ($\alpha = 0.1$), VRA achieves $\mathscr{M}_2 = 10.36$ compared to the FRA value 11.44, actually achieving slightly better resource stability and also better temporal stability. This seemingly paradoxical result arises because with very tight deadlines, even FRA must make substantial adjustments, and VRA optimization framework finds more efficient adjustment patterns. For the Extra model, the results show a dramatic loss in the stability of the starting time and a moderate increase in the stability of the resource allocation. While intuitive, this heuristic performs poorly on actual stability metrics with $\mathscr{M}_1$ values 4 to 15 times higher than optimized approaches.

2.2 Managerial insights

Having established the computational performance of VRA and FRA approaches across diverse project contexts, we now synthesize the findings with the aim of providing guidance for project managers. We will separately discuss different characteristics that impact stability. Highly interconnected project networks (many dependencies between activities) are inherently more difficult to stabilize. During project planning phases, managers should explicitly evaluate network complexity as a risk factor. For unavoidably complex networks, organizations should bargain higher α values to compensate for reduced stability potential. The relationship between resource availability and stability is complex and non-monotonic. Counterintuitively, moderately constrained resource situations can be more difficult to stabilize than either scarce or abundant resource cases. At both extremes-very scarce resources and abundant resources-stability is more easily achieved, albeit for different reasons. In scarce resource environments, the limited options create natural stability (fewer alternatives to consider). In abundant environments, resource conflicts are rare. For projects with high resource factor (activities requiring multiple resource types), resource stability challenges persist regardless of deadline. In such projects it is paramount to carefully negotiate resource flexibility arrangements with suppliers/subcontractors, trying also to reduce multi-resource requirements and complex resource interdependencies whenever possible. Moreover, maintaining buffer stocks of critical resources (if possible) would help reducing reallocation necessity.

References

Deblaere F., Demeulemeester E., Herroelen W., Van de Vonder S., 2007, "Proactive Resource Allocation Heuristics for Robust Project Scheduling", *KU Leuven Working Paper*, SSRN.

Artigues C., Michelon P., Reusser S., 2003, "Insertion techniques for static and dynamic resource-constrained project scheduling", *European Journal of Operational Research*, Vol. 149, No. 2, pp. 249–267.

Bendotti, P., Chretienne, P., Fouilhoux, P., Pass-Lanneau, A., 2023, "The anchor-robust project scheduling problem", *Operations Research*, Vol. 71, No. 6, pp. 2267–2290

On solution representations for two-stage single-machine robust scheduling

Louis Rivière[1] and Christian Artigues[2]

[1] Université Côte d'Azur, France
riviere@i3s.unice.fr
[2] LAAS-CNRS, Université de Toulouse, CNRS, France
artigues@laas.fr

Keywords: two-stage robust scheduling, single machine, solution representation

1 Introduction

We consider a two-stage robust scheduling problem under a set of discrete scenarios Ω on a single machine with release dates, precedence constraints and the objective is to minimize the maximum sum of job completion time across all scenarios. A discrete scenario $s \in \Omega$ gives for each job $j \in \mathcal{J}$ a release date r_{js} and a processing time p_{js}. In a deterministic setting the problem is denoted $1|r_i, prec|\sum C_i$ and is already strongly NP-hard, even without precedence constraints (Lenstra *et al.* 1977).

In classical single-machine robust two-stage scheduling (Aloulou and Della Croce 2008, Kouvelis and Yu 1997) with discrete scenarios, the first-stage (also called "here and now") decisions that have to be taken before the uncertainty is revealed, i.e. before one of the scenario is realized, are a total order of the jobs, in other words a job sequence. Hence, when uncertainty is progressively revealed in real time, the second-stage ("wait-and-see") decisions fix the job start time as early as possible w.r.t. the prescribed sequence. This is the earliest schedule policy (ES), which is optimal given a sequence for regular objectives. Let Θ denote the set of job sequences compatible with the precedence constraints. Let $ES(\theta, s)$ the job start times obtained with an earliest schedule policy given a job sequence $\theta \in \Theta$. The classical (two-stage) robust scheduling problem can be written $\min_{\theta\in\Theta} \max_{s\in\Omega} \sum_{j\in\mathcal{J}}(ES_j(\theta, s) + p_{js})$. One of the issue with such an approach is its lack of flexibility. Because of the restriction to follow sequence θ, the optimal sequence θ^*_s of a particular scenario s is likely to be excluded. To deal with this issue, a previous work aimed at increasing flexibility of first-stage decisions by only prescribing a partial order taking the form of a sequence of groups of permutable jobs (Rivière *et al.* 2023). The second-stage scheduling policy scans the group sequence and schedules at each decision time t (i.e. the end of the release of a job) the unscheduled task of minimal realized passed release date (FIFO policy). For a given scenario set, this solution representation scheme dominates the job sequence, because a job sequence is itself a group sequence and the FIFO policy reduces to the ES policy in this case. However, the optimal minmax group sequence may be harder to find than the optimal minmax job sequence. Furthermore, for a practical usage, the obtained first-stage decision must be compared on unseen scenario, where the dominance is not guaranteed anymore.

But these particular partial ordering of the set of jobs are only one kind of partial decision that can be made in the first stage. Indeed, any representation of a set of sequence can be seen as a partial decision, or a restriction of the set of feasible solutions. The scheduling literature is rich with examples of such representation.: partial orders, groups of permutable jobs, PQR trees, etc. Each representation type has different properties such

as how they are able to restrict the solution space, how compact the representations are, or how easily they can be handled.

In this work, we are interested in studying how the type of decision taken in the first stage affects the quality of the solutions obtained in out-of-the-box scenarios. That is, how different ways to split the decision process between first stage and second stage allows to make use of both the cheaper cost of computing time in the first stage and of the additional information available in the second stage.

2 The two-stage robust scheduling problem

A strategy F is defined by the it's first stage partial decision space Φ^F and it's second stage policy H^F. We require the second stage policy to be **valid**, **non anticipatory**, **fast**, and **order-inducing** with regard to the partial decision space. For each scenario s, given a partial decision $\pi \in \Phi^F$ and a scenario s, H^F must induce a total order $<_s^F$ on the set $\Theta(\pi)$ of represented sequences. For example, for the FIFO second stage policy and the partial decision space made of sequences of groups of permutable operations, consider 4 jobs, group sequence $\pi = \{A, B\}, \{C, D\}$ and 2 release date scenarios $r_{.1} = (0, 1, 4, 5)$ and $r_{.2} = (1, 0, 4, 3)$. Sequence ordering gives $(A, B, C, D) <_1^F (A, B, D, C) <_1^F < (B, A, C, D) <_1^F < (B, A, D, C)$ and the reverse order for scenario 2 and order $<_2^F$. This means that if scenario 1 occurs, the second-stage policy selects sequence (A, B, C, D) while sequence (B, A, D, C) is selected if scenario 2 occurs. Hence, for every partial decision π representing a set of sequences $\Theta(\pi)$, and for every scenario s, the policy H^F should be able to yield the feasible sequence $H^F(\pi, s) = \min_{<_s^F} \Theta(\pi)$ in polynomial time (the corresponding optimal earliest schedule can then be deduced). Given a strategy $F = (\Phi^F, H^F)$, the general formulation of the problem is:

$$\min_{\pi \in \Phi^F} \max_{s \in \Omega} \sum_{j \in \mathcal{J}} (ES_j(H^F(\pi, s), s) + p_{js})$$

That is, the goal is to find the restriction $\pi \in \Phi^F$ that, when used to guide the policy H^F in scheduling the set of jobs, minimizes the robust score over the set of scenarios.
We call those restrictions **metasolutions**, and each metasolution represents a set of sequences, the ones that are compatible with the partial decision. Recall that for a given sequence, the optimal corresponding schedule is the earliest schedule (ES) for regular objectives. Hence it is enough for the policy H^F to yield an admissible sequence.

3 Solution strategies

This paper aims at comparing 4 strategies F. For the first one ($F = JSEQ$), a metasolution π is a full job sequence (i.e. $\Phi^F = \Theta$ and $H^F(\pi, s) = \pi$), which resorts to the standard robust scheduling approach. For the second one ($F = GSEQ$), a metasolution π is a sequence of groups of permutable jobs and the second-stage policy is FIFO. In (Rivière *et al.* 2023), it is shown that using a starting JSEQ solution, a greedy heuristic like the one introduced in (Esswein 2023) is competitive with more complex approaches and metaheuristics. Esswein's greedy procedure starts from a sequence of single job groups and successively merges adjacent groups until no non-worsening merges exist. The two last strategies are set strategies, namely $F = SJSEQ$ and $F = SGSEQ$. Let $F = (\Phi^F, H^F)$ be a strategy, we define the set strategy $SF = (\Phi^{SF}, H^{SF})$ where metasolutions are represented by sets of metasolutions from the strategy $F : \Pi = \{\pi_1, \pi_2 \dots \pi_k\}$ with $\pi_i \in \Phi^F$. The decision space is then $\Phi^{SF} = \mathcal{P}(\Phi^F)$. In the second stage, the policy H^{SF} yields the sequence according to the order defined by $H^F : H^{SF}(\Pi, s) = \min_{<_s^F} \{H^F(\pi, s),\ \forall \pi \in \Pi\}$.

4 A polynomial algorithm for best metasolution subset extraction

Given a set of metasolutions Π, we introduce a polynomial algorithm (in the size of the set $|\Pi|$) to find a subset $\Pi^* \subseteq \Pi$ optimal with regards to the described two stage decision.

Algorithm 1 Best-Subset algorithm

1: From a starting set of metasolution Π
2: $\Pi^* = \Pi$
3: $B^* = \max_s \gamma(H(\Pi, s))$
4: **while** $|\Pi| > 0$ **do**
5: $\quad \underline{s} = \operatorname{argmax}_s \gamma(H(\Pi, s), s)$
6: $\quad \underline{\pi} = \min_{<_{\underline{s}}} \Pi$
7: $\quad \Pi = \Pi \setminus \{\underline{\pi}\}$
8: $\quad$ **if** $\max_s \gamma(H(\Pi, s), s) < B^*$ **then**
9: $\quad\quad \Pi^* = \Pi$
10: $\quad\quad B^* = \max_s \gamma(H(\Pi, s), s)$
11: Return Π^*

Algorithm 1 starts with setting the best solution so far Π^* and its corresponding score B^* by evaluating the full set of metasolutions Π with objective γ and policy H (2-3). Then, while the set of metasolutions isn't empty, the algorithm extracts the limiting scenario $\underline{s}$ and removes the corresponding metasolution $\underline{\pi} \in \Pi$ from the set (5-7). The new amputed solution is evaluated and the best found solution so far is updated if relevant (8-10). The algorithm then returns the best solution found so far Π^*.
The time complexity of the algorithm as presented is $O(|\Omega||\Pi|^2)$.

Figure 1 illustrates how the algorithm behaves at a given step. In this exemple, the algorithm aims to find the best SJSEQ subsolution from a starting JSEQ solution for an instance with 3 training scenarios. The figure displays, for each scenario, the score reached for each sequence in the JSEQ (sequences are identified by colors) in the order given by $<_s^\pi$ (hence, the leftmost values represent the score of the sequence yielded by the policy. In the first loop, the worst scenario $\underline{s} = s_3$ is extracted with $\gamma(H(BAC, s_3)) = 6$. Sequence BAC is removed, and the new worst score becomes 5 in scenario 2. In this example the reached subset $\{ACB, ABC\}$ is optimal, but note that the subset $\{ACB\}$ would be too.

	$<_s$		
$s_1 : r_A < r_B < r_C$	4	5	~~2~~
$s_2 : r_C < r_B < r_A$	~~1~~	5	6
$s_3 : r_B < r_A < r_C$	~~0~~	3	2

Fig. 1. Illustration of the best subset algorithm($|\Omega| = 3$, $\Pi = \{ACB, ABC, BAC\}$)

Proposition 1. *Given a valid policy H, and an objective γ, and a set strategy solution Π Best-Subset(Π) yields $\Pi^* = \operatorname{argmin}_{X \subseteq \Pi} \max_{s \in \Omega} \gamma(H(X, s), s)$ in polytime in $|\Pi|$.*

5 Experiments and preliminary results

Given the chain of dominance established (of respectively GSEQ over JSEQ, and SGSEQ over GSEQ), and as a proof of concept for the proposed family of strategies, we propose a "get better solution quick" scheme, that is guaranteed to improve solutions in theory. Starting from a good quality JSEQ π_{JSEQ}, an SJSEQ solution is created by generating a diverse set of JSEQ solutions including π_1. Esswein's procedure is applied to each JSEQ solution and the resulting GSEQ solutions make up an SGSEQ solution. Among these GSEQ solutions, the best one is set aside as π_{GSEQ}. The best GSEQ subset Π^*_{SGSEQ} is then extracted using the best subset algorithm.

Method	$\|\Omega\| = 5$			$\|\Omega\| = 25$			$\|\Omega\| = 100$		
Δ	0.1	0.3	0.5	0.1	0.3	0.5	0.1	0.3	0.5
JSEQ	1.029	1.103	1.196	1.047	1.188	1.348	1.079	1.261	1.506
GSEQ	1.028	1.101	1.124	1.042	1.097	1.135	1.061	1.128	1.154
SGSEQ	1.023	1.022	1.017	1.041	1.083	1.088	1.061	1.127	1.153

Table 1. Performance (Score/Bound)

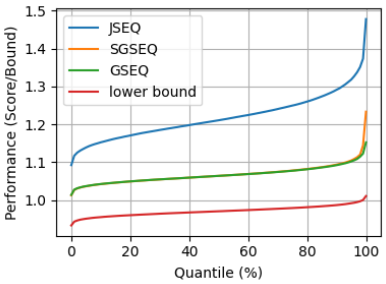


Table 2. Testing scenarios distribution ($|\Omega| = 100$)

Instances are generated starting from a deterministic $1|r_i, prec|\_$ instance where durations of jobs p_i are selected from a uniform distribution in $[50, 100]$ and release dates in a $[0, \sum p_i/2]$ interval. Precedence constraints are generated randomly with a 1% density.
From this deterministic instance, scenarios are generated by sampling new release dates r_i^s around the nominal release date r_i based on a normal law $\mathcal{N}(r_i, r_i * \Delta)$, where Δ is a variability parameter of the generated instances. The higher the value of Δ, the more different each sampled scenario will be. The sampled scenario set is then divided into training scenarios and testing scenarios.
We compare the performance of the different computed solutions (π_{JSEQ}, π_{GSEQ}, and Π^*_{SGSEQ} described earlier) as a ratio of the score to a bound of the best possible strategy score on the instance (see (Riviere 2023)).

Preliminary results suggest that, as expected, the proposed dominating strategies outperform the dominated ones in training. Although, for the GSEQ and SGSEQ strategies, when the number of training scenario increases, the necessity for robustness reduces the difference in training score to almost none, indicating that the gain observed in training on fewer sampled scenarios likely represents overfitting (Table 1). Indeed, we can observe on Figure 2 that in the testing phase , the SGSEQ method is actually competitive with the GSEQ method only when the number of training scenarios is high.
Overall, while the theoretical and practical dominance of GSEQ solutions over JSEQ ones was demonstrated, the preliminary results suggest that the increase in expressivity provided by the SGSEQ strategy isn't necessary to provide even better solutions on these instances. Further research is required to characterize what instances would benefit the theoretical superiority of SGSEQ.

References

Mohamed Ali Aloulou and Federico Della Croce, 2008, "Complexity of single machine scheduling problems under scenario-based uncertainty", *Operations Research Letters* , vol. 36(3), pp. 338–342.

Panos Kouvelis and Gang Yu, 1997, "Robust scheduling problems", Chapter 7 in *Robust Discrete Optimization and Its Applications* , Springer.

Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan and Peter Brucker, 1977, "Complexity of machine scheduling problems", *Annals of Discrete Mathematics*, Vol. 1, pp.343–362, 1977.

Louis Rivière, Christian Artigues and Hélène Fargier, 2023, "Two-stage stochastic/robust scheduling based on permutable operation groups", *Annals of Operations Research*, Vol. 332, pp. 645–687.

Carl Esswein, 2003, "Un apport de flexibilité séquentielle pour l'ordonnancement robuste", PhD Thesis, Iniversity of Tours, url=http://www.theses.fr/2003TOUR4022.

Louis Riviere, 2023, "Representation compacte d'ensembles de solutions pour l'ordonnancement sous incertitudes", PhD Thesis, Universite Paul Sabatier - Toulouse III, url=http://www.theses.fr/2003TOUR4022.

Multi-mode time-constrained project scheduling in environments subject to disturbances

R. Gaouar[1], V. Borodin[1], A. Dolgui[1], J.-P. Lemaitre[2], B. Poyet[2], S. Thevenin[1]

[1] IMT Atlantique, LS2N-CNRS, Nantes, FRANCE
{rim-djazia.gaouar, valeria.borodin, alexandre.dolgui, simon.thevenin}@imt-atlantique.fr
[2] Airbus Atlantic, Colomiers, FRANCE
{jean-philippe.lemaitre, baptiste.b.poyet}@airbus.com

Keywords: Aerospace manufacturing, Multiple modes, Scheduling, Uncertainty, Stress-testing, Target cycle time, Stressed cycle time

1 Introduction

Aerospace assembly systems face many sources of uncertainty (e.g., resource shortages, variable task durations) making it challenging to rely on a single baseline plan. In daily operations, decision-makers must constantly deal with limited resources and inefficient use of those resources. They also need to support repeated allocation and reallocation decisions subject to industrial constraints, often under multiple criteria such as workload balance and cost.

To the best of our knowledge, only limited research has addressed how to model and manage uncertainty on aircraft assembly lines. Pulido et al. (2017) provide a detailed representation of operational uncertainties within a discrete-event simulation of an aircraft assembly line. In addition to planned maintenance and scheduled absences, the model explicitly accounts for four types of stochastic disturbances: *steady throughout the year:* machine failures and errors that require extra work, *variable over the year:* operator absences, and final test rejections. These uncertainty factors are used to assess the impact of troublemakers on delivery performance and to evaluate alternative mitigation strategies, such as adding workers or implementing night shifts. Ziarnetzky et al. (2014) model structural and supply-chain uncertainties in low-volume mixed-model assembly lines, including random supplier delays, variable replenishment lead times, finite buffer capacities, and stochastic troublemakers in upstream processes. The way uncertainty is understood and described as a risk strongly influences its analysis and, therefore, may have significant implications for both risk management and decision-making (Aven 2016).

In this work, we focus on multi-mode time-constrained project scheduling in aircraft assembly systems subject to operator-related troublemakers. Starting from a fixed baseline sequence and a predefined Target Cycle Time (TCT), we evaluate the sequence performance across a set of representative TCT-detractor scenarios and characterize the resulting distribution of stressed cycle times. Rather than optimizing under uncertainty, the approach analyzes how the baseline behaves when subjected to troublemakers. In addition, key statistical characteristics of this distribution are translated into solution-quality indicators such as reliability, stability, and robustness.

2 Problem statement and solution approach

The scheduling of aircraft assembly activities exhibits characteristics that differ considerably from those of classical high-volume, high-automation production systems. Aircraft assembly stations operate with long cycle times, multi-manned operations, limited buffer

capacities, strong precedence and spatial constraints, and intensive human labor. Because tasks can be executed in different ways (e.g., with different operator profiles, team sizes, or processing times), the corresponding scheduling problem takes the form of a Multi-Mode Resource-Constrained Project Scheduling Problem (MMRCPSP). In MMRCPSP, each task can be executed in one of several modes, each specifying its processing time, required resources, and feasible areas. MMRCPSP also involves precedence relations and task incompatibility. Time lags restrict the minimum or maximum delay between related tasks.

In this study, we focus specifically on operator absences and derive a set of representative scenarios from real-life settings. Operator absences are modeled as extra time units added to the reference processing times of tasks. Since the problem is multi-mode, each task can be executed using different combinations of operators, which enables us to capture how operator availability affects task durations and overall schedule feasibility.

To evaluate the quality of a baseline sequence (i.e., the order in which tasks are performed on assigned resources) against disturbed processing times, we employ a stress-testing procedure that perturbs the processing times of tasks according to a set of TCT-detractor scenarios Ω. We start from a baseline sequence S^*, determined using reference processing times and a given Target Cycle Time (TCT). We then stress-test S^* by evaluating its performance over Ω. For each scenario $\omega \in \Omega$, we compute the resulting cycle time $TCT_(S^*, \omega)$ and collect these values into a set that approximates the distribution of stress-tested cycle times. Based on the resulting empirical distribution, we derive the service level (or on-time completion), i.e., the probability that the stressed cycle time remains below or equal to TCT.

3 Numerical experiments

For illustration purposes, numerical experiments were conducted on a subset of benchmark instances for the deterministic time-constrained multi-mode scheduling problem provided in (Borreguero Sanchidrian et al. 2024). As indicated in Table 1, each instance includes 8 tasks and is characterized by a defined number of precedence constraints, time lags, incompatibility constraints, renewable resources, working areas, execution modes, and a fixed cycle time. After assuming that the deterministic processing times correspond to the reference values, a baseline sequence was computed using CP Optimizer 22.1.2, by solving the model proposed in (Borreguero Sanchidrian et al. 2024).

Table 1. Benchmark instance characteristics

Instance	Tasks	Precedence constraints	Time lags	Incompat. constraints	Ressources	Areas	Modes	TCT
Set1-8	8	6	1	1	2	2	11	350
Set3-8	8	7	1	1	2	2	17	1,650
Set4-8	8	7	1	1	2	2	15	1,400

Fig. 1 illustrates how the cycle time is affected under uncertainty and shows differences in sensitivity when operator $R1$, operator $R2$, or both $R1/R2$ are absent. To assess the impact of stochastic processing times on the performance of a given sequence, we evaluate the resulting distribution of the random cycle time T^*.

Since a full probability distribution is difficult to compare across sequences, we summarize its key features using a set of statistical aggregators. Table 2 provides the definition of each aggregator together with its interpretation.

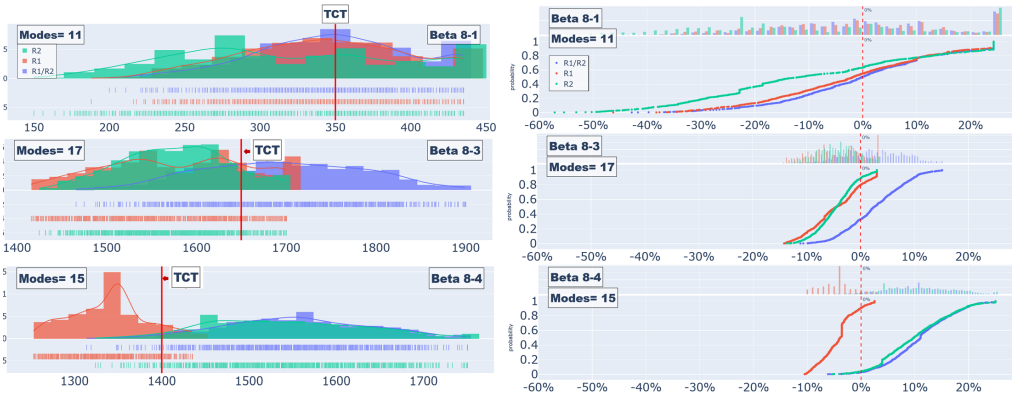
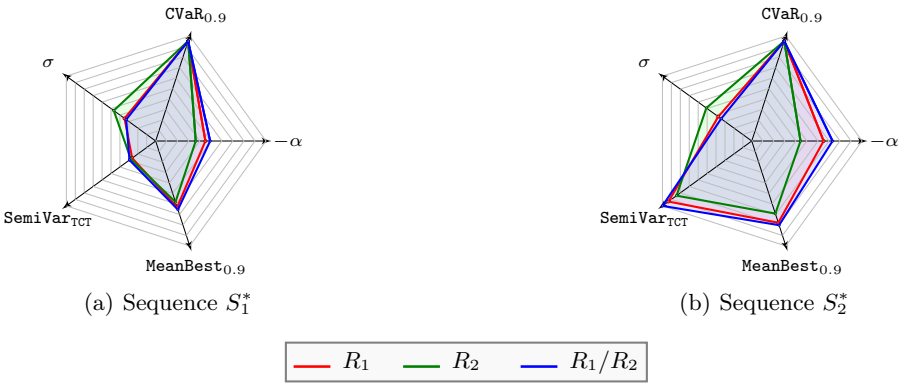


Fig. 1. Example of simulated distributions of stressed cycle times versus the target cycle time: Instance `Set-8-1`, Instance `Set-8-3`, Instance `Set-8-4`.

Table 2. Statistical aggregators for a sequence with random cycle time T^*, where TCT denotes the target cycle time. The up/down arrow symbol denotes the optimization sign (max/min).

Agregator	Definition	Interpretation
$\uparrow \alpha$	$\mathbb{P}(T^* \leq \text{TCT})$	*Reliability:* Probability that the realized performance meets TCT
$\downarrow \sigma$	$\sqrt{\mathbb{V}\text{ar}[T^*]}$	*Stability:* Variability around the mean, lower σ indicates more stable and predictable performance.
$\downarrow \text{MeanBest}_{0.9}$	$\mathbb{E}[T^* \mid T^* \leq q_{0.9}]$	*Performance on normal/good conditions:* Conditional mean over the best 90% of realizations
$\downarrow \text{SemiVar}_{\text{TCT}}$	$\mathbb{E}[(T^* - \text{TCT})_+^2]$	*Downside robustness:* Downside semi-variance relative to TCT, focus only late/abnormal cases
$\downarrow \text{CVaR}_{0.9}$	$\mathbb{E}[T^* \mid T^* \geq q_{0.9}]$	*Robustness to rare abnormal conditions:* Conditional Value-at-Risk (CVaR) at level 0.9, average performance in the worst 10% of scenarios



(a) Sequence S_1^* (b) Sequence S_2^*

Fig. 2. Comparison of statistical behavior of two equivalent deterministic sequences S_1^* and S_2^* under uncertainties in terms of statistical aggregators defined in Table 2: Instance `Set-8-1`.

For illustration purposes, let us focus on Instance `Set-8-1` and consider two equivalent sequences S_1^* and S_2^* (i.e., having identical sum of maximum operator profile demands).

Fig. 2 summarizes the key characteristics of the distributions shown in Fig. 1 through scalar aggregators that capture average behavior, variability, reliability with respect to TCT, and robustness to atypical operating conditions.

4 Conclusions

This study focuses on multi-mode time-constrained project schedules in aircraft assembly systems that are subject to operator-related troublemakers. Starting from a deterministic baseline sequence, we quantify the impact of representative detractor scenarios on cycle-time performance. The resulting stressed cycle-time distribution is then analyzed using complementary statistical aggregators. The proposed indicators provide a multidimensional view of solution performance under uncertainty, capturing reliability, stability, robustness, and sensitivity to abnormal events.

Acknowledgements

The present work was conducted in the framework of the project ACCURATE (Achieving Resilience through Manufacturing as a Service, Digital Twins and Ecosystems), supported by European Union's Horizon Europe research and innovation program under grant agreement No. 101138269, HORIZON-CL4-2023-TWIN-TRANSITION-01-07. However, the views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or European Health and Digital Executive Agency (HADEA). Neither the European Union nor the granting authority can be held responsible for them.

The authors would like to thank Tamara Borreguero of Airbus Defence and Space, Getafe (Madrid), for providing the sets of small- and medium-sized instances used in this paper.

References

Aven, T., 2016, "Risk Assessment and Risk Management: Review of Recent Advances on their Foundation", *European Journal of Operational Research*, Vol. 253, No. 1, pp. 1–13.

Bierbusse J., L. Monch and A. Biele, 2025, "Heuristic Approaches for a Multi-Mode Resource Availability Cost Problem in Aircraft Manufacturing", *Computers & Operations Research*, Vol. 176, p. 106888.

Borreguero Sanchidrian T., T. Portoleau, C. Artigues, A. Garcia Sanchez, M. Ortega Mier and P. Lopez, 2024, "Large neighborhood search for an aeronautical assembly line time-constrained scheduling problem with multiple modes and a resource leveling objective", *Annals of Operations Research*, Vol. 338, No. 1, pp. 13–40.

Borreguero T., F. Mas, J. L. Menendez and M. A. Barreda, 2015, "Enhanced Assembly Line Balancing and Scheduling Methodology for the Aeronautical Industry", *Procedia Engineering*, Vol. 132, pp. 990–997.

Pulido R., T. Borreguero-Sanchidrian, A. Garcia-Sanchez and M. Ortega-Mier, 2017, "Analysis of the robustness of production scheduling in aeronautical manufacturing using simulation: a case study", in *IFIP International Conference on Product Lifecycle Management*, pp. 174–183.

Ziarnetzky T., L. Monch and A. Biele, 2014, "Simulation of Low-Volume Mixed Model Assembly Lines: Modeling Aspects and Case Study", *Proceedings of the 2014 Winter Simulation Conference*, pp. 2101–2112.

J - Constraint programming #3

Periodic Scheduling of Grouped Time-Triggered Signals on a Single Resource

Josef Grus[1,2], Zdenek Hanzalek[2] and Claire Hanen[3,4]

[1] DCE, FEE, Czech Technical University in Prague
[2] IID, CIIRC, Czech Technical University in Prague
[3] Sorbonne University, CNRS, LIP6
[4] University Paris Nanterre
josef.grus@cvut.cz, zdenek.hanzalek@cvut.cz, claire.hanen@lip6.fr

Keywords: bin packing, periodic scheduling, signal grouping

1 Grouping of Time-Triggered Signals

Time-triggered messages are of crucial importance in modern communication networks. Offline-generated schedules, which specify start times for periodic messages, enable us to achieve deterministic behavior in critical applications. In automotive and avionics domains, so-called signals (measurements and commands) are periodically generated and communicated (via messages) among sensors, controllers, and actuators.

However, the message contains not only the useful signal data, but also necessary metadata, e.g., message ID. Metadata is stored as a header or tail and extends the message size; when the signal is very short (as it often is in applications), sending each in a separate message is inefficient. Thus, several signals are grouped into a single message, depending on their periodicity and length, and sent with just one header. Such an approach increases the utilization of the communication resource (link or bus), since less bandwidth is wasted on headers (Kuaban et al. 2021). However, grouping the signals into messages is complicated. The maximum size of the message (including the metadata) is finite, since longer messages have a lower probability of successful delivery. Also, longer messages are less flexible for scheduling in a periodic setting. This is similar to the work of Huan et al. (2019), where the compromise between energy efficiency and latency for IoT devices was investigated.

In this paper, we study the fundamental problem of grouping time-triggered signals into messages and periodic scheduling of messages on a single resource.

2 Problem Statement

The joint problem of grouping signals and scheduling messages is modeled as an extension of the Periodic Scheduling Problem (PSP), which is generally strongly NP-complete (Cai & Kong 1996). Our problem consists of set of tasks $\mathcal{J} = \{J_1, \ldots, J_i, \ldots, J_n\}$. Each task models one of the time-triggered signals. J_i is associated with integer period T_{α_i} and integer processing time p_i. We consider a set of harmonic periods, which is often used in applications; for any $J_i, J_{i'}$, T_{α_i} is an integer multiple of $T_{\alpha_{i'}}$, or vice versa. This implies period set $\mathcal{T} = \{T_0, \ldots, T_u, \ldots, T_{r-1}\}$ of size r, such that $\forall k \in \{1, \ldots, r-1\}, T_u = b_u T_{u-1}, b_u \in \mathbb{Z}^+$. Note that harmonic periods allow us to utilize efficient packing model, based on canonical form outlined later. α_i maps J_i to one of the r periods, partitioning $\mathcal{J}$ to task-period sets $\mathcal{J}_u = \{J_i : T_{\alpha_i} = T_u\}$.

We allow grouping of signals with the same period only. Thus, tasks of $\mathcal{J}_u$ are grouped into groups (i.e., messages) given by set $\mathcal{G}_u = \{G_{u,1}, \ldots G_{u,j}, \ldots, G_{u,|\mathcal{J}_u|}\}$; sufficient number of groups is used to accommodate single-task-per-group corner case. Group $G_{u,j}$ has

period T_u, and its processing time depends on tasks assigned to it. With a set of tasks $\mathcal{J}_{u,j}$ assigned to $G_{u,j}$, we define the group size $gs_{u,j}$ as:

$$gs_{u,j} = \begin{cases} hs + \sum_{J_i \in \mathcal{J}_{u,j}} p_i & |\mathcal{J}_{u,j}| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Thus, $gs_{u,j}$ includes the header of constant size hs, if there are any tasks. The group size is also upper-bounded by a fixed maximum group size MG. Every task is part of some group: $\mathcal{J}_u = \bigcup_{\forall G_{u,j}} \mathcal{J}_{u,j}$. The groups are then scheduled on a resource. Our grouping differs from grouping in the context of a multi-processor scheduling (Ren et al. 2024), where task sets are partitioned among multiple resources. As in Grus. et al. (2024) the schedule can be visualized by stacking each of the first $\frac{T_{r-1}}{T_0}$ intervals of length T_0 on top of another. A row k is then called an observation interval and corresponds to the interval $[k-1 \cdot T_0, k \cdot T_0)$. Each group of period T_u is scheduled periodically in $\frac{T_{r-1}}{T_u}$ such intervals. The shorter the period of the group is, the more occurrences of it are present. This is shown in Fig. 1a

As was applied in Lukasiewycz et al. (2009), Eisenbrand et al. (2010), and proven in Grus. et al. (2024), only solutions in canonical form need to be considered. Canonical form means that each task/group of a given period has all the tasks/groups with a shorter period to its left in each observation interval. Thus, scheduling groups into suitable observation intervals can be modeled similarly to bin packing. It is only necessary to decide in which observation interval the first occurrence of the group is scheduled. The indices of intervals relevant for the first occurrence are given by the set $\mathcal{B}_u = \{0, \ldots, T_u/T_0 - 1\}$.

To find a feasible solution, each observation interval utilization (sum of processing time of occurrences of present groups) needs to be limited by the shortest period T_0. However, this constraint can be made into a soft constraint by minimizing the $C_{\max}$, the maximum utilization of any observation interval. If $C_{\max} \leq T_0$, the solution is feasible with respect to the minimum period. Any margin $T_0 - C_{\max}$ makes the schedule more flexible, e.g., for scheduling additional sporadic tasks.

3 Mathematical Programming Model

In this section, we describe the mixed-integer linear programming (MILP) model for the extended PSP outlined in the previous section. The model assigns task J_i to exactly one group $G_{u,j}$ via binary variables x_{uij} in Eq. (3). Group size $gs_{u,j}$ is calculated and constrained with Eqs. (4) to (6), where the header is included via binary variable z_{uj}.

$$\min C_{\max} \tag{2}$$

$$\sum_{\forall G_{u,j} \in \mathcal{G}_u} x_{uij} = 1 \qquad \forall T_u \in \mathcal{T}\ \forall J_i \in \mathcal{J}_u \tag{3}$$

$$\sum_{\forall J_i \in \mathcal{J}_u} p_i \cdot x_{uij} \leq |\mathcal{J}_u| \cdot z_{uj} \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u \tag{4}$$

$$gs_{u,j} = hs \cdot z_{uj} + \sum_{\forall J_i \in \mathcal{J}_u} p_i \cdot x_{uij} \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u \tag{5}$$

$$gs_{u,j} \leq MG \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u \tag{6}$$

$$\sum_{k \in \mathcal{B}_u} y_{ujk} = 1 \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u \tag{7}$$

$$c_{ujk} \leq gs_{u,j} \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u\ \forall k \in \mathcal{B}_u \tag{8}$$

$$c_{ujk} \leq T_0 \cdot y_{ujk} \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u\ \forall k \in \mathcal{B}_u \tag{9}$$

$$c_{ujk} \geq gs_{u,j} - T_0 \cdot (1 - y_{ujk}) \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u\ \forall k \in \mathcal{B}_u \tag{10}$$

$$p_{uk} = \sum_{\forall G_{u,j} \in \mathcal{G}_u} c_{ujk} \qquad \forall T_u \in \mathcal{T}\ \forall k \in \mathcal{B}_u \tag{11}$$

$$\sum_{\forall T_u \in \mathcal{T}} p_{u[k \mod |\mathcal{B}_u|]} \leq C_{\max} \qquad \forall k \in \mathcal{B}_{r-1} \tag{12}$$

$$C_{\max} \in \mathbb{Z},\ p_{uk} \in \mathbb{Z} \qquad \forall T_u \in \mathcal{T}\ \forall k \in \mathcal{B}_u \tag{13}$$

$$x_{uij} \in \{0,1\} \qquad \forall T_u \in \mathcal{T}\ \forall J_i \in \mathcal{J}_u\ \forall G_{u,j} \in \mathcal{G}_u \tag{14}$$

$$z_{uj} \in \{0,1\},\ gs_{u,j} \in \mathbb{Z} \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u \tag{15}$$

$$y_{ujk} \in \{0,1\},\ c_{ujk} \in \mathbb{Z} \qquad \forall T_u \in \mathcal{T}\ \forall G_{u,j} \in \mathcal{G}_u\ \forall k \in \mathcal{B}_u \tag{16}$$

Each group is scheduled in exactly one suitable observation interval via y_{ujk}, see Eq. (7). Even empty groups are scheduled, but with $G_{u,j} = 0$, they do not affect the solution. The contribution of group $G_{u,j}$ to utilization of the observation interval k is tracked via c_{ujk} variables. Due to the dynamic group size, this is done with a big-M approach in Eqs. (8) to (10); $c_{ujk} = gs_{u,j}$ if $y_{ujk}1$, and zero otherwise. Finally, individual contributions are combined to obtain the load p_{uk} of the given observation interval by groups with period T_u in Eq. (11), which, summed across periods, forms $C_{\max}$ (13).

Note that a simpler model, based on Lukasiewycz et al. (2009), Grus. et al. (2024) can be utilized for two special cases: when each task is in an individual group (obtaining the upper bound on $C_{\max}$). Only one group is necessary for any number of tasks with period T_u in the observation interval k (obtaining the lower bound on $C_{\max}$).

4 Experiments

We performed experiments with several synthetic instances, derived from those in Grus. et al. (2024). We varied parameter values of hs and MG, and we obtained 47 example instances with 50 - 600 tasks and up to six distinct periods. We compared three solvers (with five minutes time limit): Gurobi v10.0, CP-SAT v9.14 of OR-Tools, and CP Optimizer v22.1. We were mostly interested in comparing MILP and CP solvers, since in our previous work (Grus. et al. 2024), CP Optimizer was slightly better than Gurobi.

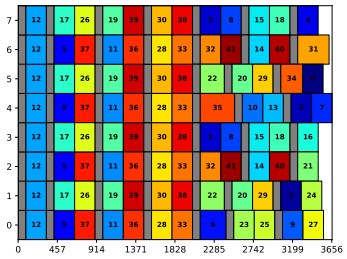
	# feasible	mean bg	median bg	mean rank
Gurobi	47	0.10	0.00	1.15
CP-SAT	42	6.13	1.17	2.13
CP Optimizer	40	1.90	0.00	1.72

Table 1: Performance of the solvers for 47 instances. Best gap bg [%] is gap between method's $C_{\max}$ and best found $C_{\max}$. Rank is calculated for each instance.

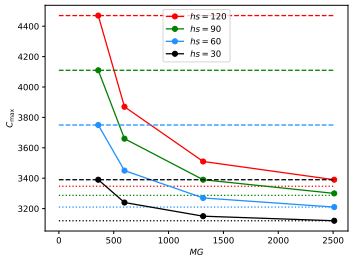
As is shown in Table 1, the Gurobi MILP solver was able to slightly outperform the CP-SAT solver of OR-Tools and CP Optimizer. It was able to find a feasible solution for each instance, and it reported the smallest averaged best gap bg and the averaged rank of the methods. However, the actual MIP gap was zero only for the less complex instances; for the harder instances, the MIP gap after five minutes was often more than 10 %.

In Fig. 1, several results of the Gurobi MILP solver are presented. In Fig. 1a, we can see how the maximum group size and headers affect the schedule. The header of size 90 is required for groups with a maximum size of 600. Headers are shown as gray rectangles at the start of the block of tasks implicitly included in the group.

In Fig. 1b we can see how the increase of the maximum group size MG improves the objective $C_{\max}$. Furthermore, the effect of different header sizes hs is quite predictable, essentially moving the curves upwards with steeper gradients. Finally, we can see the upper and lower bound solutions with dashed and dotted lines, respectively.



(a) Example solution. Header size was set to 90 and maximum group size to 600. Headers are shown as gray tasks at the start of each (implicit) group.



(b) Parameter sensitivity analysis showing effect of hs and MG on $C_{\max}$. Dashed lines are solutions obtained for the upper-bound model, and dotted lines for the lower-bound model.

Fig. 1: Example solution and parameter sensitivity analysis for Gurobi solver on instance with $\mathcal{T} = \{4000 \cdot i \mid \forall i \in \{1, 2, 4, 8\}\}$. Each solution was optimized for 5 minutes.

5 Conclusion

It is clear that grouping of signals offers significant advantages in real-life communication settings by improving resource utilization. We solved the problem using MILP and CP solvers. In the future, we will investigate the multi-resource setting (Grus et al. 2025) and the usage of different values of MG per period. Furthermore, we will study special cases, including instances with divisible group sizes and periods for which the second-level problem is polynomial.

Acknowledgments

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS25/144/OHK3/3T/13 and co-funded by the European Union under the project ROBOPROX (reg. no. CZ.02.01.01/00/22_008/0004590).

References

Cai, Y. & Kong, M. (1996), 'Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems', *Algorithmica* **15**(6), 572–599.

Eisenbrand, F. et al. (2010), 'Solving an avionics real-time scheduling problem by advanced IP-methods', *European Symposium on Algorithms* pp. 11–22.

Grus., J., Hanen., C. & Hanzalek., Z. (2024), Packing-inspired algorithms for periodic scheduling problems with harmonic periods, *in* 'Proc. of the 13th ICORES', SciTePress, pp. 101–112.

Grus, J., Hanen, C. & Hanzalek, Z. (2025), 'Periodic chains scheduling on dedicated resources - a crucial problem in time-sensitive networks', *Computers & Operations Research* **180**, 107072.

Huan, X., Kim, K. S., Lee, S. & Kim, M. K. (2019), 'Optimal message bundling with delay and synchronization constraints in wireless sensor networks', *Sensors* **19**(18).

Kuaban, G. S., Atmaca, T., Kamli, A., Czacorski, T. & Czekalski, P. (2021), 'Performance analysis of packet aggregation mechanisms and their applications in access (e.g., iot, 4g/5g), core, and data centre networks', *Sensors* **21**(11).

Lukasiewycz, M. et al. (2009), FlexRay schedule optimization of the static segment, *in* 'Proc. of IEEE/ACM CODES+ISSS 2009', pp. 363–372.

Ren, J., Zhang, J., Li, X., Cao, W., Li, S., Chu, W. & Song, C. (2024), 'Enhanced harmonic partitioned scheduling of periodic Real-Time tasks based on slack analysis', *Sensors* **24**(17).

A Constraint Satisfaction Formulation for the CTMC Representation of Stochastic Flow Shops

Ziyue Xu, Marcello Urgo and Lei Liu

[1] Department of Mathematical Sciences, University of Nottingham, Ningbo, China
daisy.ziyue0526@outlook.com
[2] Department of Mechanical Engineering, Politecnico di Milano, Milano, Italy
marcello.urgo@polimi.it
[3] Nottingham University Business School China, University of Nottingham, Ningbo, China
lei.liu@nottingham.edu.cn

1 Introduction and Problem Statement

Modern production and service systems exhibit inherent, unpredictable variability in processing times. These fluctuations stem from diverse sources, including tool wear and degradation, operator absenteeism, network congestion, unplanned maintenance, and resource contention. While deterministic scheduling models offer computationally tractable and often elegant solutions in theory, their practical utility is frequently compromised: minor stochastic disturbances readily propagate through the system, and low-probability, high-impact delays can disproportionately govern the overall performance. We consider a stochastic m-machine permutation flow shop, a set of n jobs must be processed sequentially on each of m machines, adhering to the same fixed machine order. The processing time $p_{i,j}$ of job i on machine j is modeled as a non-negative random variable, which is considered general and may vary for each job-machine pair (i, j). Upon selection of a fixed job sequence, $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, the system makespan, $C_{\max}(\mathbf{x})$, becomes a random variable. The cumulative distribution function (CDF) of $C_{\max}(\mathbf{x})$ is dictated by the joint statistics of all processing times. [Kulkarni and Adlakha, 1986] showed that stochastic activity networks can be modeled as Markov-regenerative processes, enabling exact estimation of makespan distributions, and state generation process can be performed using the uniformly directed cut (UDC) approach ([Creemers et al., 2010]), while the UDC approach often suffers from generating many equivalent or symmetric states, requiring frequent merging of duplicates, which slows down computation significantly. Hence, we propose a constraint satisfaction–based framework that builds the CTMC state space for stochastic flow shops more efficiently, allowing exact and scalable evaluation of the makespan distribution under general processing time models.

2 Methodology

Although the exponential distribution is frequently used in stochastic scheduling due to its memoryless property, it often poorly represents real-world processing time variability. To overcome this limitation, we adopt phase-type (PH) distributions, which can approximate any non-negative distribution arbitrarily well while remaining analytically tractable. In a stochastic flow shop, the system evolves from the start of the first job on first machine until the completion of the last job on machine m. At any time t, each operation is in exactly one of three states: *Completed* (0), *Running* (1), or *Pending* (2), the latter indicating that the operation is waiting for its upstream job or preceding job on the same machine to finish. Given a fixed job sequence $\{1, 2, \ldots, n\}$, the execution progresses through a series of stochastic transitions: starting with only (J_1M_1) running, the system moves to states where multiple operations may be active, with future states depending on which running operation completes first. This process continues until the absorbing state, where all operations are completed, is reached. Although the resulting state space can become large and highly branched due to concurrency and precedence constraints, we observe that the state-generation mechanism can be precisely modeled as a constraint satisfaction problem (CSP), with variables encoding operation statuses and constraints enforcing flow-shop sequencing rules.

The Constraint Satisfaction Problem (CSP) formulation systematically defines the feasible state space of the CTMC by identifying all valid combinations of operation statuses across the n jobs and m machines that adhere to the intrinsic operational rules of a permutation flow shop. The state of the system is defined by the status of the $n \times m$ operations. The decision variable $x_{i,j}$ indicates the status of operation (i, j) (job i on machine j), where

$$x_{ij} = \begin{cases} 0, & \text{operation } x_{ij} \text{ has been completed} \\ 1, & \text{operation } x_{ij} \text{ is running in this state} \\ 2, & \text{operation } x_{ij} \text{ is waiting} \end{cases} \tag{1}$$

S.t.

$$x_{ij} \geq x_{i,j-1}, \quad i \in 1, ..., n, \quad j \in 2, ..., m \tag{2}$$

$$x_{ij} \geq x_{i-1,j}, \quad i \in 2, ..., n, \quad j \in 1, ..., m \tag{3}$$

$$\sum_{j} (x_{ij} = 1) \leq 1, \forall j \tag{4}$$

$$\sum_{i} (x_{ij} = 1) \leq 1, \forall i \tag{5}$$

$$\sum_{i,j} (x_{ij} = 1) \geq 1, \tag{6}$$

$$x_{1,j} = 2 \Rightarrow x_{1,j-1} \neq 0, \forall j \geq 2 \tag{7}$$

$$x_{i,1} = 2 \Rightarrow x_{i-1,1} \neq 0, \quad \forall i \geq 2 \tag{8}$$

$$x_{i,j-1} = 0 \quad \&\& \quad x_{i-1,j} = 0 \Rightarrow x_{i,j} \neq 2, \quad \forall i \geq 2, \forall j \geq 2 \tag{9}$$

Constraint (2) ensures job operations follow their machine sequence. Constraint (3) maintains the permutation property on each machine. Constraint (4) and (5) limit each job and each machine to at most one running operation, respectively. Constraint (6) ensures at least one operation is running. Constraint (7) prevents the first job from having a pending operation if its predecessor is complete. Constraint (8) prevents a pending operation on the first machine if the previous job's operation is complete, and Constraint (9) forces an operation to start if both its job and machine predecessors are complete.

Once the complete and feasible state space $\mathcal{S}$ is generated by the CSP model, the system's evolution is described by the Continuous-Time Markov Chain (CTMC). State transitions, from a starting state $\mathbf{X}_k \in \mathcal{S}$ to a subsequent state $\mathbf{X}_l \in \mathcal{S}$, are initiated by the single, time-homogenous event of one operation completion. These transitions are governed by the physical irreversibility of the processing status:

- No operation may revert from a *Completed* status to a *Running* status.
- No operation may bypass the active processing stage, meaning a direct transition from Pending to *Completed* is prohibited.
- The transition event must correspond to the completion of exactly one operation, which transitions from *Running* to *Completed*.

The infinitesimal generator matrix Q of the CTMC is constructed as a block matrix partitioned according to the states in $\mathcal{S}$. The diagonal block matrices encode the internal phase transitions of the Running operations within each state, while the off-diagonal transition matrices encode the rates of movement between states $\mathbf{X}_k$ and $\mathbf{X}_l$. Both the diagonal and off-diagonal blocks can be calculated efficiently through Kronecker algebra operations, which applied also to the subgenerator matrices of the Phase-Type (PH) distributed processing times [Liu and Urgo, 2023]. The makespan $C_{\max}$ is defined as the time required to reach the absorbing state in which all operations have completed. Its cumulative distribution function, denoted $F_{C_{\max}}(t)$, admits a phase-type representation, characterized by an initial probability row vector $\boldsymbol{\beta}$ and a subgenerator matrix $\mathbf{Q}$ obtained by removing the absorbing state, i.e., by eliminating the last row and column of the CTMC's infinitesimal generator. The CDF is then given analytically by $F_{C_{\max}}(t) = 1 - \boldsymbol{\beta} e^{\mathbf{Q}t}\mathbf{e}$, where $\mathbf{e}$ is a column vector of ones.

3 Results

A set of test instances was generated with $n = 5, 10, 15, 20$ jobs and $m = 2, 5, 10$ machines. Processing times were modeled using phase-type distributions, randomly generated via the BuTools library by specifying the mean and the number of phases. The experimental results, summarized in Table 1 and Fig. 1, show how our constraint satisfaction based state-space generation model scales with

problem size. Average CPU times ranged from 0.01 s to 1658.78 s, and a clear power-law relationship emerged between the number of generated states and the state-generation time, highlighting the efficiency and structured nature of our approach. Moreover, the procedure is anytime-capable: if a time limit is reached, it returns the subset of states generated so far, enabling a controlled approximation of the full state space when exact enumeration becomes infeasible. For comparison, we also implemented the UDC method on the same instances. Even for a modest case with $n = 5$ jobs and $m = 5$ machines, UDC required about 1 minute of computation time, and this quickly grew to over 1 hour as the problem size increased. This comparison further underscores the scalability advantage of our CSP-based framework.

$$S(n,m) = \sum_{k=1}^{m} \binom{n+k-1}{k} = \sum_{k=1}^{n} \binom{m+k-1}{k} \tag{10}$$

n	**m**	**No. State**	**Avg. CPU time (s)**
5	2	20	0.01
5	3	55	0.01
5	5	251	0.05
5	10	3002	0.68
10	2	65	0.01
10	3	285	0.05
10	5	3002	0.59
10	10	184755	60.20
15	2	135	0.03
15	5	15503	3.42
15	10	3268759	1658.78
20	2	230	0.05
20	5	53129	14.49

Table 1. State size and CPU time.

References

[Creemers et al., 2010] Creemers, S., Leus, R., and Lambrecht, M. (2010). Scheduling markovian pert networks to maximize the net present value. *Operations Research Letters*, 38(1):51–56.

[Kulkarni and Adlakha, 1986] Kulkarni, V. G. and Adlakha, V. G. (1986). Markov and markov-regenerative pert networks. *Operations Research*, 34(5):769–781.

[Liu and Urgo, 2023] Liu, L. and Urgo, M. (2023). A branch-and-bound approach to minimise the value-at-risk of the makespan in a stochastic two-machine flow shop. *International Journal of Production Research*.

A two-stage approach for radiotherapy scheduling

Hugues Rauwel[1,2], Christian Artigues[2], Romain Guillaume[1] and Laure Vieillevigne[3]

[1] IRIT-CNRS, Université de Toulouse, Toulouse, France
[2] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[3] Institut Universitaire du Cancer de Toulouse-Oncopole, Département de Physique Médicale, Toulouse, France

Keywords: Healthcare, Scheduling, Stochastic Optimization.

1 Introduction

Radiotherapy is a cancer treatment technique used in over 50% of cancer cases, either independently or in combination with surgery and/or chemotherapy. It involves delivering controlled doses of ionizing radiation to a specific area of the body to destroy cancerous cells while sparing surrounding healthy tissue as much as possible. The duration of a radiotherapy treatment varies between patients due to the site (tumor location) and treatment technique. Radiotherapy treatments are delivered by linear accelerators, referred to as "linacs", where the patient receives, in general, a fractional daily dose over several weeks.

Cancer treatment centers face a challenging problem when it comes to radiotherapy sessions scheduling, due to various medical and technical constraints as well as resource availability. In many centers, such as in our cancer treatment center partner, scheduling is done manually by the staff. Optimizing the process could lead to better quality of treatment by sparing unnecessary waiting times, since it has been shown that it was related to more negative outcomes.

Moreover, uncertainty in patient arrival is a major concern of cancer treatment centers since it could put the radiotherapy service to stress, consequently decreasing the standard of care. Once patients agreed to radiotherapy and have gone through their first stage of care, they wait for a variable amount of time to start the irradiation sessions. In compliance with our application case, we model this patient arrival uncertainty as an uncertainty on patients' release dates. The idea is to take into account all available knowledge, certain and uncertain, in order to achieve a better overall schedule with the ambition of a clinical usage.

In this work, we address the radiotherapy scheduling problem with uncertain patient arrivals.

2 Problem Statement

2.1 Deterministic setup

Let $\mathcal{P}$ be the set of patients ($p \in \mathcal{P}$) to be scheduled in an horizon $\mathcal{T}$. Assuming perfect knowledge of arrival over $\mathcal{T}$, the schedule of all patients could be computed at decision time $t = 0$. However, $\mathcal{P}$ is revealed throughout the time horizon $\mathcal{T}$, therefore we have at a given time $t \in \mathcal{T}$, $\mathcal{P}(t)$ as the set of known patients to be scheduled, with $\mathcal{P} = \cup_{t \in \mathcal{T}} \mathcal{P}(t)$.

This means that patients in $\mathcal{P}(t) \setminus \cup_{t' < t} \mathcal{P}(t')$ are still unknown at any days $t' < t$, and their schedule on a local scheduling horizon $\mathcal{L}_t$ must be fixed before day $t + 1$ without possibility to change it later due to the communication of the schedule to the patients at day t. The local scheduling horizon $\mathcal{L}_t$ is the set of days such that $\mathcal{L}_t = \{t, \ldots, t + D\}$, where D is a constant value, large enough to ensure all patients' sessions can be scheduled in this local horizon.

Two major aspects can influence the performance of this batch scheduling procedure: the model used for scheduling and the scheduling period. Indeed, too-frequent batch scheduling leads to excessive myopic "online" scheduling , but not too-frequent batch scheduling leads to delays for patients. In this work, we will focus on analyzing the performance of the batch scheduling model where scheduling frequency is set by the cancer treatment center.

We define $\mathcal{K}$ as the set of treatment machines ($k \in \mathcal{K}$) and $\mathcal{K}^p \subseteq \mathcal{K}$ as the set of authorized machines where patient p can effectively be treated. For each patient p, we denote by I_p the number of sessions also called fractions, by b_p the treatment frequency (1 for daily sessions and 2 for one session every other day) and by u_p the emergency of patient p (1 for very urgent, 2 for urgent and 3 for normal urgency). We also define a_p as the admission date (in our case, the scanner date) and m_p as the medical validation date. Finally, the ready date of patient p is denoted r_p and d_p is the first session due date for patient p. In this section, we assume that we know all the attributes of all the patients that need to be scheduled.

The decisions are the start dates for each patient p : $\mathbf{S}_p = (S_p^1, \ldots, S_p^{I_p})$ and their associated treatment machine. A schedule x_t is the set of the decisions variables above for every $p \in \mathcal{P}(t)$. The objectives are to minimize the sum of waiting time $S_p^1 - r_p$ and the sum of treatments spans $S_p^{I_p} - S_p^1$ to softly enforce treatment frequency. For concision's sake, if x_t is a schedule, we will simply denote by $f(x_t)$ this objective value.

Illustrative example

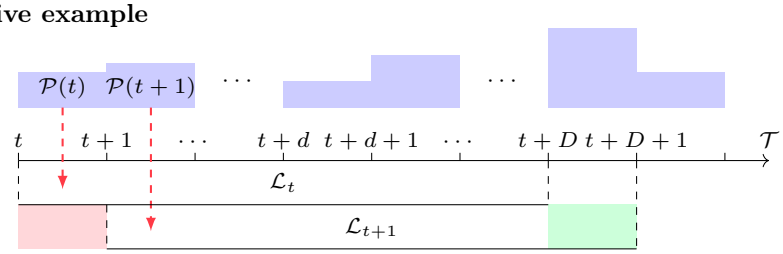
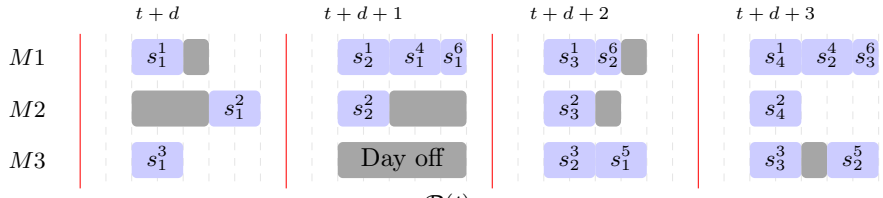


Fig. 1: Dynamic scheduling process.



$\mathcal{P}(t)$

p	r_p	$\mathcal{K}^p$	p	r_p	$\mathcal{K}^p$
p_1	$t+d$	M1	p_4	$t+d+1$	M1,M3
p_2	$t+d$	M1,M2	p_5	$t+d+2$	M1,M2,M3
p_3	$t+d$	M3	p_6	$t+d+1$	M1,M2

Fig. 2: Scheduling process at instant t.

Figure 1 shows that at a given time t patients $\mathcal{P}(t)$ are scheduled in the local horizon $\mathcal{L}_t$. As the scheduling process moves forward in time, $\mathcal{L}_t$ becomes $\mathcal{L}_{t+1}$ by removing the first day of $\mathcal{L}_t$ and adding an extra day. Decisions taken at time $t' < t$ become data in our

current problem.
In figure 2, days of the local scheduling horizon $\mathcal{L}_t$ are separated by vertical red lines and the interval between each dotted vertical line represents a 3-hour time slot. The sessions' durations as well as machines operating time ranges have been exaggerated for the sake of the example. Gray tasks are planned unavailability or already scheduled sessions at time $t' < t-1$. The notation s_k^p stands for the session k of patient p. The table from figure 2 gives the release dates r_p and the set of authorized machines $\mathcal{K}^p$ for patients p in the batch $\mathcal{P}(t)$. The release dates are values in $\mathcal{L}_t$ of the form $t+d+x$ where d is the minimal treatment setup delay and x varies depending on the treatment type.

2.2 Leveraging uncertainty

In practice, we remark that the knowledge we dispose at time $t \in \mathcal{T}$ can be divided in two groups :

- The patients with certain information, ready to be scheduled, $p \in \mathcal{P}^C(t)$.
- The patients with partial information, waiting for more knowledge to be included in the scheduling batch, $p \in \mathcal{P}^I(t)$.

Previously, we considered that $\mathcal{P}(t) = \mathcal{P}^C(t)$ also called "myopic" scheduling, while we now have $\forall t \in \mathcal{T}$, $\mathcal{P}(t) = \mathcal{P}^C(t) \cup \mathcal{P}^I(t)$ and the sets are two-by-two disjoints.

More precisely, the available information for a patient $p \in \mathcal{P}^I(t)$ is all above-described patient characteristics except patient ready dates without known their precise probability distribution since in practice we know only the minimum and maximum validation times. Note that patient due dates are set after the scanner step and are not subject to uncertainty.

The new scheduling strategy is a batch scheduling strategy, while leveraging the additional uncertain knowledge to better account for new arrivals.

We propose to address this problem with a two-stage model, where first-stage scheduling is done using second-stage knowledge. Let $\mathcal{X}_t$[4] be the set of all possible schedules for certain patients ($p \in \mathcal{P}^C(t)$) and Ξ_t the uncertainty set. We also define $\mathcal{Y}_{x,\xi}$ as the set of all possible schedules for uncertain patients ($p \in \mathcal{P}^I(t)$) relatively to $x \in \mathcal{X}_t$ and $\xi \in \Xi_t$. The two-stage problem can be written :

$$\min_{x \in \mathcal{X}_t} f(x) + \bigoplus_{\xi \in \Xi_t} g(x, \xi) \tag{1}$$

with $g(x,\xi) = \min_{y \in \mathcal{Y}_{x,\xi}} f(y) + h(y)$.
f is a real-value function defined over $\mathcal{X}_t$ and $\mathcal{Y}$ and $h : \mathcal{Y} \to \mathbb{R}$ is a specific cost of scheduling of uncertain patients. $\bigoplus$ is an aggregation operator over the uncertainty set.
If $\bigoplus = \max$, one can recognize a classical robust 2-stage model while $\bigoplus = \mathbb{E}$ leads to a stochastic 2-stage model.

We chose to model the uncertainty set Ξ with a scenario-based approach, where scenarios are variations on uncertain patients' release dates. The uncertainty set Ξ is a discrete set and will be denoted $\mathcal{S} = \{(r^1_{p_1}, \cdots, r^1_{p_{|\mathcal{P}^I(t)|}}), \cdots, (r^{|\mathcal{S}|}_{p_1}, \cdots, r^{|\mathcal{S}|}_{p_{|\mathcal{P}^I(t)|}})\}$.

3 Results

We measured the cumulative objective values over the horizon $\mathcal{T} : \sum_{t \in \mathcal{T}} f(x_t^m)$ where x_t^m are first stage solutions computed with method m that can be "myopic" or 2-stage with limited number of scenarios ($|\mathcal{S}| = \{1, 2, 5, 10\}$).

[4] $x \in \mathcal{X}_t$ is a set of s_k^p defined in the example.

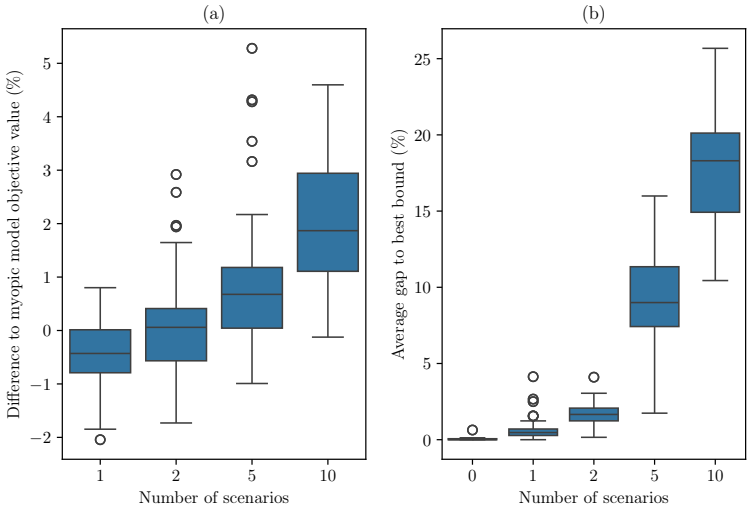


Fig. 3: Results for (a) Differences to cumulative "myopic" objective value over $\mathcal{T}$: $\sum_{t\in\mathcal{T}} f(x_t^{|\mathcal{S}|}) - f(x_t^{myopic})/\sum_{t\in\mathcal{T}} f(x_t^{myopic})$ and (b) solver best bound, relatively to the number of scenarios.

We implemented the models defined above using constraint programming (CP-Sat) and evaluated them through simulations using real-world data. Our approach demonstrated a clear advantage over the myopic batch scheduling method. Notably, the two-stage model effectively reduced patient waiting times, a critical factor in improving the quality of care. We also explored the impact of the number of scenarios on the model's performance, finding that while increasing scenarios captures more uncertainty, it also increases computational complexity. Our results indicate that a single-scenario approach or using a small number of representative scenarios can achieve a favorable balance between solution quality and computational efficiency. This is inline with the results obtained in (Frimodig *et al.* 2023), except that they considered only one scenario while we show that including more scenarios may bring additional improvements and we also have a much finer scheduling granularity thanks to the constraint programming model.

References

Vieira, Bruno and Hans, Erwin W and van Vliet-Vroegindeweij, Corine and van de Kamer, Jeroen and van Harten, Wim", "Operations research for resource planning and -use in radiotherapy: a literature review", *BMC Med Inform Decis Mak, 2016.*

Frimodig, Sara and Enqvist Per and Kronqvist Jan, "A Column Generation Approach for Radiation Therapy Patient Scheduling with Planned Machine Unavailability and Uncertain Future Arrivals", *arXiv, 2023.*

Frimodig, Sara and Schulte, Christian, "Models for Radiation Therapy Patient Scheduling", *Principles and Practice of Constraint Programming, 2019.*

K - Resource-constrained project scheduling

Using a relax-and-fix approach to solve the stochastic resource-constrained project scheduling problem with flexible resource profiles

Ann-Kathrin Mendl and Julia Rieck

University of Hildesheim, Universitätsplatz 1, Germany,
Operations Research Group, Institute for Business Administration and Information Systems
{mendl;rieck}@bwl.uni-hildesheim.de

Keywords: Stochastic resource-constrained project scheduling, Flexible resource profiles, Fix-and-optimize, Sample average approximation

1 Introduction and problem description

Project work plays a central role in many companies, particularly in research and development, IT, and finance. For our purposes, we represent projects as activity-on-node networks $N = (V, E)$. Set $V = \{0, \ldots, n+1\}$ contains all non-preemptive activities, including the two fictitious activities 0 and $n+1$, and set E comprises the precedence constraints between activities $i, j \in V$. If the objective is to minimize the project duration, the resulting problem is the classical *resource-constrained project scheduling problem* (RCPSP), which assumes a fixed duration $d_i \geq 0$ for each activity $i \in V$ and constant resource requirements $r_{ik} \geq 0$ for all renewable resources $k \in K$ throughout the execution of activity i. Since these assumptions ignore uncertainties regarding future durations and resource requirements, a resulting solution (*schedule*) $S = (S_i)_{i \in V}$, including the start times S_i of activities i, may be too restrictive and unsuitable for the underlying application. The *stochastic RCPSP* (SRCPSP), introduced by Möhring et al. (1984), addresses some of these uncertainties by modeling d_i as random variables while keeping r_{ik} constant. In many practical settings, however, only the total resource requirement (*workload*) $w_{ik} \geq 0$ of activity $i \in V$ on resource $k \in K$ is known in advance, and the realized duration of i depends on how the resources are allocated over time. This motivates the *flexible RCPSP* (FRCPSP) of Kolisch et al. (2003), which allows for a flexible distribution of workloads across the individual time periods $t \in T$.

In our previous work (cf. Mendl and Rieck, 2026), we combined the SRCPSP with the FRCPSP and introduced the *SRCPSP with flexible resource profiles* (FSRCPSP). In the corresponding model, each activity $i \in V$ is associated with a stochastic workload, which is obtained by scaling the deterministic workload w_{ik}^{d} with an activity-specific factor $f_i > 0$ drawn from an underlying distribution $\mathbf{P}$. In this way, both the general structure of resource demands and the ratio between the individual resource requirements of activity i are preserved while uncertainty is introduced. The flexible resource allocation over time is modeled through real-valued decision variables $r_{ikt} \geq 0,\ i \in V, k \in K, t \in T$, which must lie within the predefined lower $\underline{r}_{ik}$ and upper limits $\overline{r}_{ik}$. An activity is completed once its workload is exactly covered, i. e., $\sum_{t \in T} r_{ikt} = w_{ik}$ holds for all i and k. The model also includes binary decision variables; x_{it} indicates whether activity i is started at time t or not, while y_{it} takes the value 1 if activity i has not yet been completed at time t, and 0 otherwise. By connecting the two decision variables x_{it} and y_{it}, it can be determined whether activity i is in progress at time t or not. In order to ensure feasibility under uncertainty, we formulate the precedence and resource constraints as joint chance-constraints by specifying a *confidence level* (CL). Since the resulting chance-constrained model cannot be solved directly, it was reformulated using the *sample average*

approximation (SAA). In the resulting SAA-FSRCPSP model, the decision variables d_i^π, r_{ikt}^π, and y_{it}^π depend on the scenarios $\pi \in \mathcal{S}$. Taking into account a sample-based CL $(1-\epsilon)$ with $\epsilon = 0.1$, the number of violated scenarios is limited to $\lfloor |\mathcal{S}| \cdot \epsilon \rfloor$.

In what follows, we intend to solve the SAA-FSRCPSP with a *relax-and-fix* (R&F) approach. To this end, related work is presented in Sect. 2.

2 Relax-and-fix and fix-and-optimize for scheduling problems

Relax-and-fix and fix-and-optimize (F&O) approaches have been applied to a wide range of scheduling problems (for a survey, see, e. g., Tanksale and Jha, 2020). F&O methods date back to Pochet and Wolsey (2006), who introduced an iterative approach in which a sequence of mixed-integer programs is solved. In each iteration, the binary decision variables are decomposed into two sets; one set of decision variables is fixed, while the other set is optimized together with all continuous decision variables. A formal framework for F&O was later provided by Helber and Sahling (2010), who distinguished between product-, resource-, and process-oriented decomposition strategies for the multi-level capacitated lot-sizing problem (CLSP). Stadtler and Heinrichs (2024) combined sample average approximation with a product-oriented F&O heuristic for the stochastic CLSP and introduced an additional postprocessing step in which the scenario-based model is solved as a linear program with a larger scenario set.

In order to solve the resource renting problem, Reinke and Zimmermann (2022) applied a F&O procedure. Furthermore, for resource-constrained project scheduling problems, mainly R&F methods have been developed, in which an additional set of binary decision variables is relaxed in each iteration so that they can take values between 0 and 1 (i. e., from the interval $[0, 1]$). Etminaniesfahani et al. (2024) proposed a time-oriented R&F method for the multi-mode RCPSP based on rolling time windows. Vasilyev et al. (2023) introduced a two-phase approach for the multi-skill RCPSP that combined a R&F procedure with a large neighborhood search. Bigler et al. (2024) presented a unit-oriented R&F matheuristic for the multi-site RCPSP, in which sequencing decisions are fixed independently of activity start times, enabling stable subsequences while retaining temporal scheduling flexibility. The review of the related work shows that R&F and F&O methods have already been applied to the RCPSP and its variants. However, no corresponding approach has been proposed to address the stochastic or flexible RCPSP. In the following Sect. 3, we therefore introduce a specific R&F procedure to solve our SAA-FSRCPSP.

3 Activity-oriented fix-and-optimize for the SAA-FSRCPSP

In our *activity-oriented R&F* procedure, we transfer the decomposition concept of the unit-oriented R&F approach by Franz et al. (2019) to the activity-based structure of the SAA-FSRCPSP. In a first step, an initial solution S is generated in a greedy manner using a serial schedule generation scheme with the priority rule "earliest start time first".

The subsequent iterative phase concentrates on localized adjustments to activity start times while preserving all precedence and resource constraints of the underlying SAA-FSRCPSP. For every activity $i \in V$ or for every activity set V_a, with $a = 1, \ldots, \beta$ and $\bigcup_{a=1}^{\beta} V_a = V$, we restrict the range of start times to a discrete time window $\widetilde{W}_i$, where $\widetilde{W}_i = \{\max\{ES_i, S_i - \omega\}, \ldots, \min\{LS_i, S_i + \omega\}\}$ and ω determines the refinement around the start time S_i; ES_i and LS_i specify the earliest and latest start time of i. All decision variables x_{it} which are not in $\widetilde{W}_i$ are fixed to zero, resulting in a subproblem that includes fewer binary decision variables, contains additional real-valued decision variables, and can be solved more efficiently. Depending on the number of activities and resources, a time

limit $t_{\text{lim}} \in \{10, 30, 60, 120, 180, 240\}$ in seconds is set. The solution is schedule S', which is only accepted if it satisfies CL $= (1 - \epsilon)$ and provides an improved makespan S'_{n+1}. Otherwise, the procedure considers another ω for activity (set) i or fixes the start times taking into account the greedy solution S.

Algorithm 1 Procedure of the proposed activity-oriented R&F

```
Determine a feasible, greedy solution S by applying a serial schedule generation scheme
for i = 0 to |V| − 1 (or i = V_1 to V_{β=3}) do
    Set counter σ = 0 and set upper limit σ̄ = 2
    while limit σ̄ on attempts σ is not reached do
        Choose parameter ω = rand(1, 10) and set σ := σ + 1
        Define W̃_i = {max{ES_i, S_i − ω}, ..., min{LS_i, S_i + ω}}
        Fix x_it := 0 for all t ∉ W̃_i
        Solve subproblem of the SAA-FSRCPSP with t_lim using (a) x_it ∈ {0,1}, ∀t ∈ W̃_i,
        (b) x_jt ∈ [0,1], ∀j > i, t ∈ T, and (c) all already fixed decision variables
        if solution S' of the subproblem is feasible then
            Fix x_{i,S'_i} := 1 and x_it := 0 for all t ∈ T \ S'_i; set counter σ := σ̄
            if S' improves the makespan then Update S = S' with S_{n+1} = S'_{n+1}
        else if σ = σ̄ then Fix x_{i,S_i} := 1 and x_it := 0 for all t ∈ T \ S_i
return Solution S
```

Algorithm 1 is implemented in Python using the CPLEX API. In Sect. 4, we evaluate its performance and compare the results with optimal solutions and our previously developed scatter search (implemented in C++, cf. Mendl and Rieck, 2026).

4 Computational results and outlook

Based on the PSPLIB library, we generated 400 instances with $n = \{10, 30\}$ real activities. Since the PSPLIB does not provide single-mode RCPSP instances with $n = 10$, we derived them from the multi-mode dataset (j10 files) by fixing the first execution mode of each activity. We always selected the first 50 instances from the library and adapted each of them to the resource configurations $|K| = \{1, 2, 3, 4\}$, resulting in 200 instances for each n. The deterministic workloads are defined as $w^{\text{d}}_{ik} = r^{\text{d}}_{ik} \cdot d^{\text{d}}_i$ and serve as the expected workloads $\mathbf{E}(w_{ik})$. The stochastic workloads follow $w^{\pi}_{ik} = f_i \cdot \mathbf{E}(w_{ik})$ with $f_i \sim \beta\,[0.5, 2.25]$ as in Lamas and Demeulemeester (2016). To enable flexible resource allocation, we set the upper limit to $\overline{r}_{ik} = r^{\text{d}}_{ik}$ and the lower limit to $\underline{r}_{ik} = \lceil 0.5 \cdot \overline{r}_{ik} \rceil$. The maximum runtime of each approach is limited to $T_{\max} = 7{,}200$ seconds. Moreover, we set the number of scenarios (*sample size*) to $|\mathcal{S}| = 10$ and the CL to $(1 - \epsilon) = 0.9$.

Table 1 reports the results obtained by R&F and our previously developed scatter search. For each configuration, we present the average project duration $\varnothing_{S_{n+1}}$ and the average runtime $\varnothing_{t_{\text{cpu}}}$ in seconds. The GAP-values measure the deviation of the heuristic solutions from the optimal results achieved by GAMS 39.3 and CPLEX 22.1. For each instance, the GAP (in percentage) is computed individually, and the arithmetic mean $\varnothing_{\text{GAP}}$ is included. The results obtained show that the proposed R&F procedure provides a strong performance for all instances. While $\varnothing_{\text{GAP}}$ ranges between 0 % and 4 %, the runtimes remain relatively low for all instances. Overall, the results confirm that the R&F approach outperforms the scatter search in terms of solution quality.

Building on this, the examination of additional decomposition strategies is planned in the future. For example, a time- (cf. Franz et al., 2019) or scenario-based (cf. Stadtler and Heinrichs, 2024) approach could be suitable for our SAA-FSRCPSP. Moreover, larger instances and larger sample sizes should also be investigated.

Table 1. Comparison of Opt, R&F, and scatter search approaches on instances with $n = \{10, 30\}$ activities, setting CL = 0.9 and sample size $|\mathcal{S}| = 10$, grouped by $|K|$

n	$\|K\|$	Opt		R&F			Scatter Search		
		$\varnothing_{S_{n+1}}$	$\varnothing_{t_{\text{cpu}}}$	$\varnothing_{S_{n+1}}$	$\varnothing_{t_{\text{cpu}}}$	$\varnothing_{\text{GAP}}$	$\varnothing_{S_{n+1}}$	$\varnothing_{t_{\text{cpu}}}$	$\varnothing_{\text{GAP}}$
10	1	28.84	47.65	28.84	33.53	0.00	30.16	18.43	5.01
	2	29.34	132.07	30.12	38.68	2.75	30.50	25.79	4.28
	3	29.56	177.62	29.92	66.99	1.05	30.78	37.42	4.82
	4	30.15	479.50	31.00	138.23	3.74	31.26	33.59	4.18
30	1	87.08	274.37	87.12	142.28	0.06	92.76	100.77	7.68
	2	87.06	266.39	87.06	171.69	0.00	91.98	119.35	6.50
	3	87.26	569.36	87.48	238.18	0.27	94.44	132.76	8.99
	4	86.88	755.62	88.56	306.64	2.20	92.20	127.14	6.83

References

Bigler, T., Gnägi, M., & Trautmann, N. (2024). MIP-based solution approaches for multi-site resource-constrained project scheduling. *Ann Oper Res*, *337*, 627–647.

Etminaniesfahani, A., Gu, H., Naeni, L. M., & Salehipour, A. (2024). An efficient relax-and-solve method for the multi-mode resource constrained project scheduling problem. *Ann Oper Res*, *338*, 41–68.

Franz, A., Rieck, J., & Zimmermann, J. (2019). Fix-and-optimize procedures for solving the long-term unit commitment problem with pumped storages. *Ann Oper Res*, *274*, 241–265.

Helber, S., & Sahling, F. (2010). A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *Int J Prod Econ*, *123*, 247–256.

Kolisch, R., Meyer, K., Mohr, R., Schwindt, C., & Urmann, M. (2003). Ablaufplanung für die Leitstrukturoptimierung in der Pharmaforschung. *J Bus Econ*, *78*, 825–848.

Lamas, P., & Demeulemeester, E. (2016). A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *J Sched*, *19*(4), 409–428.

Mendl, A.-K., & Rieck, J. (2026). Sample average approximation for the stochastic resource-constrained project scheduling problem with flexible resource profiles. *Ann Oper Res*, online.

Möhring, R., Radermacher, F., & Weiss, G. (1984). Stochastic scheduling problems I: General strategies. *Math Meth Oper Res*, *28*(7), 193–260.

Pochet, Y., & Wolsey, L. A. (2006). *Production Planning by Mixed Integer Programming.* Springer, New York.

Reinke, M., & Zimmermann, J. (2022). Solving the resource renting problem with an adapted fix-and-optimize heuristic. *IEEE Proceedings, Kuala Lumpur*, 1119–1123.

Stadtler, H., & Heinrichs, N. (2024). Multi-period descriptive sampling for scenario generation applied to the stochastic capacitated lot-sizing problem. *OR Spectrum*, *46*, 639–668.

Tanksale, A., & Jha, J. K. (2020). A hybrid fix-and-optimize heuristic for integrated inventory-transportation problem in a multi-region multi-facility supply chain. *RAIRO Oper Res*, *54*, 749–782.

Vasilyev, I., Muftahov, I., & Ushakov, A. (2023). MIP heuristics for a resource constrained project scheduling problem with workload stability constraints. *MOTOR Proceedings, Jekaterinburg*, 212–223.

Genetic algorithms for scheduling projects with multi-skilled resources and training decisions

Guillaume Vermeire[1], Mario Vanhoucke[1,2,3]

[1] Faculty of Economics and Business Administration, Ghent University, Belgium
guillaume.vermeire@ugent.be,mario.vanhoucke@ugent.be
[2] Technology and Operations Management, Vlerick Business School, Belgium
[3] UCL School of Management, University College Londen, UK

Keywords: Project scheduling, Multi-skilled resources, Training.

1 Introduction

In the last decade, the resource-constrained project scheduling problem (RCPSP) has been extended in different ways to make the problem more applicable to real life. One of these extensions, called the multi-skilled resource-constrained project scheduling problem (MSRCPSP), considers the human aspect of resources. In the MSRCPSP, renewable resources are replaced by a set of human workers who are able to master a number of skills. Many solution procedures have been proposed for the MSRCPSP [Afshar-Nadjafi, 2021]. Although these algorithms generally result in high-quality solutions, they typically assume that the resources and their mastered skills remain static.

This research assumes that a project manager has received a specified budget to expand the mastered skills of its workforce. This can be done by deciding which resource-skill pairs to train, as long as the training budget is not exceeded. These choices have a large impact on the achievable project objective as training inefficient pairs will not lead to an improved schedule. Additionally, even if the budget allows extensive training, it is not necessarily the case that each additional training leads to a project schedule with a lower makespan. Investing in training that does not contribute to an objective improvement would be inefficient. For that reason, training decisions should be made sparingly.

To model this problem, the MSRCPSP is extended with training decisions (Section 2), with the budget represented as a maximum quantity of these decisions. To solve the problem, two different meta-heuristic methodologies are proposed that are able to find high quality solutions in a reasonable amount of time (Section 3). The two methodologies are compared and preliminary results are shown in Section 4.

2 Problem description

A project is represented by a topologically ordered acyclic activity-on-the-node network $G = (N, A)$, in which the set N contains all activities of the project including a dummy start activity, a dummy end activity, and $|N|-2$ non dummy

activities. The finish start relations between activities are represented by the arc set A. Each activity $i \in N$ can be scheduled by giving it a starting time s_i. Once an activity i is scheduled, it requires p_i time units to become completed. The skill requirements r_{ij} specify the integer amount of resources that must be assigned to a skill j on activity i for it to be scheduled.

The skill requirements from an activity can be performed by a set of renewable multi-skilled resources. This set is called the workforce R which contains $|R|$ resources. Each resource $k \in R$ masters one or more skills from the skill set S which is represented by the binary variables b_{kj} in the skill distribution. This variable is equal to one if resource k masters skill j and zero otherwise. Resources are assigned to activities such that each of the skill requirements r_{ij} from that activity i are satisfied. This is done by using the assignment variable x_{ijk} which equals 1 if resource k is assigned to activity i on a requirement for skill j. When a resource is assigned, it stays assigned for the full duration p_i of that activity and it can not be assigned to other activities during that period.

Training decisions are made using the binary decision variable t_{kj} which represents the training of a resource k on the skill j. This variable equals 1 if resource k is trained to master skill j, or zero otherwise. Note that a resource-skill pair $\{k, j\}$ can only be trained if the resource k does not already master the skill j ($b_{kj} = 0$). In this research, training does not require time to be completed, and is hence done before the execution of the project. Once the training decisions are implemented in the workforce R, the trained workforce (R^t) is obtained. The training quantity q represents the maximum number of training decisions that can be made, while the training used variable q^u represents the number of training decisions that are made $\sum_k \sum_j t_{kj} = q^u \leq q$

The objective is to find the minimal project makespan by making choices concerning the starting times of activities (s_i), the assignment of multi-skilled resources (x_{ijk}) and the training of resource-skill pairs (t_{kj}), taking the allowed training quantity into account (q). Note that, between two project schedules with the same makespan, the one that requires less used training (q^u) is preferred.

3 Methodologies

Two methodologies are described that each use a different approach considering the training. The first method separates the problem in two phases. The training decisions are made in the first phase after which a well performing algorithm for the MSRCPSP is used in the second phase to generate a project schedule (Section 3.1). For the second method, a specialized genetic algorithm (GA) is created in which training, assignments and scheduling are integrated in a single phase (Section 3.2). Both approaches are visualized in Figure 1.

3.1 Adapted GA

The GA from [Snauwaert and Vanhoucke, 2025], originally created to solve the MSRCPSP, is adapted for the training problem by implementing an additional

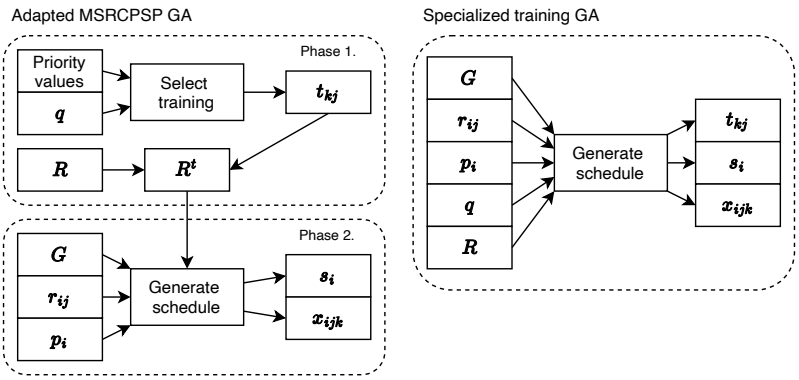


Fig. 1. Overview of procedures for the training problem

phase before the scheduling procedure. In this phase, the workforce R is converted into a trained workforce R^t by making $q^u = q$ training decisions. The number of possible, differently trained workforces corresponds to $\binom{q_{max}}{q}$, with q_{max} representing the maximum number of training options ($q_{max} = |S| \times |R| - \sum_k \sum_j b_{kj}$). As q_{max} becomes larger, the number of possible trained workforces increases quickly. Therefore, considering each possible R^t is not an effective approach to find the set of trained resource-skill pairs for which the second phase can find the lowest project makespan. In this research, one single R^t is considered by making a selection of promising training decisions. This selection is made by using a ranking of all training possibilities. This ranking can be seen as a priority list of resource-skill pairs. When a training quantity of q is considered, the first phase of this methodology selects the q highest ranking training options to be selected. The second phase generates project schedules according to the algorithm of [Snauwaert and Vanhoucke, 2025]. However, since a choice is made in the first phase, the performance of the GA in the second phase lies in the context of this choice. An additional disadvantage is that q training is imposed on the workforce, even when this training does not offer benefits in the scheduling procedure. These disadvantages are overcome with the next methodology.

3.2 Specialized GA

In this specialized one-phase GA, the core scheduling procedure is adapted such that the assignment of resources can happen when these resources do not master the required skills. This means that a resource k which does not master (or is trained upon) the skill j ($b_{kj} + t_{kj} = 0$) is allowed to be assigned on a skill requirement r_{ij} of activity i for skill j. However, each time such an assignment happens, the resource k is considered to be trained on the skill j and one training is incurred ($t_{kj} = 1$). These assignments may happen as long as $\sum_k \sum_j t_{kj} < q$. This way, training only occurs when it provides a direct advantage to schedule activities. As result, each training decision is utilized sparingly.

4 Preliminary results

Instances with a $q_{max} = 100$ are selected from the MSLIB dataset, created by [Snauwaert and Vanhoucke, 2023]. The dataset is solved for different q values. In the first approach, a MIP is created and solved with Gurobi. Additionally, both described GA's are used. The results are shown in Figure 2 in which the makespan of the obtained solutions is represented in the context of the q.

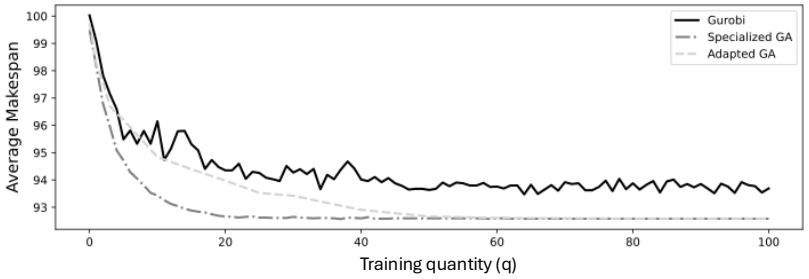


Fig. 2. Fair comparison between Gurobi and two proposed GA's.

From this figure it becomes clear that both GA's generally outperform Gurobi. The specialized GA obtains the lowest makespans on low training quantities compared to the adapted GA. When the training quantity is high, both solution procedures have the same solution quality. Deciding which skills and resources to train is a difficult and strategic decision that can have a large impact on the achievable quality of the project. This research proposes two methods for making these training decisions in a project scheduling context. The results indicate that a two-phase GA is outperformed by the specialized GA when training opportunities are scarce, which can be explained by the integration of the training procedure in the scheduling algorithm.

References

[Afshar-Nadjafi, 2021] Afshar-Nadjafi, B. (2021). Multi-skilling in scheduling problems: A review on models, methods and applications. *Computers Industrial Engineering*, 151:107004.

[Snauwaert and Vanhoucke, 2023] Snauwaert, J. and Vanhoucke, M. (2023). A classification and new benchmark instances for the multi-skilled resource-constrained project scheduling problem. *European Journal of Operational Research*, 307(1):1–19.

[Snauwaert and Vanhoucke, 2025] Snauwaert, J. and Vanhoucke, M. (2025). A solution framework for multi-skilled project scheduling problems with hierarchical skills. *Journal of Scheduling*, 28:289–310.

Formulating the Two-Stage Agile Sprint Planning Problem: Optimization and Reinforcement Learning Models

Claudio Szwarcfiter[1] and Avraham Shtub[2]

[1] Faculty of Industrial Engineering and Technology Management, Holon Institute of Technology, Israel
szwarcfiterc@hit.ac.il
[2] Faculty of Data and Decision Sciences, Technion—Israel Institute of Technology, Israel
shtub@technion.ac.il

Keywords: Agile Project Management, Reinforcement Learning, Mixed-Integer Linear Programming.

1 Introduction

Agile project management organizes work in short sprints and seeks early delivery of a minimum viable product (MVP). We study a sprint planning problem in which a finite set of issues $I = M \cup N$ is divided into must-have issues M and nice-to-have issues N. Each issue i has story points sp_i, predecessor set P_i, and required resource type req_i; if $i \in N$, completing it yields value v_i. Resource type $r = 1, \ldots, R$ has K_r instances, where instance k has per-sprint capacity cap_{rk}, cost c_{rk}, and availability bound u_{rk}. The project is constrained by a total budget B and a deadline of D sprints, and work may be split across sprints.

We propose a two-stage formulation. Stage 1 builds the MVP as early as possible at minimum cost. Stage 2 uses the remaining budget and remaining sprints to maximize the value of the nice-to-have issues. The proposed setting differs from standard project scheduling formulations in that mandatory MVP delivery and optional value-driven scope expansion are modeled as two linked decision stages. The problem combines elements of the resource-constrained project scheduling problem and the knapsack problem, and is therefore NP-hard (Blazewicz, Lenstra & Rinnooy Kan 1983, Karp 1972).

2 Two-Stage MILP Formulation

Table 1 summarizes the core notation. In addition, binary variables $start_{\sigma i}$ and $finish_{\sigma i}$ indicate whether issue i starts or finishes in sprint σ.

For sprint σ, let $x_{\sigma irk} \in [0, 1]$ denote the fraction of issue i processed in sprint σ by instance k of resource type r; thus (r, k) denotes resource type r and its kth instance. Let $y_\sigma \in \{0, 1\}$ equal 1 if sprint σ is executed, and let $q_{\sigma rk} \in \{0, 1, 2, \ldots\}$ be the number of units of instance k of type r used in sprint σ. Binary variables $start_{\sigma i}$ and $finish_{\sigma i}$ indicate whether issue i starts or finishes in sprint σ.

Stage 1: MVP development. Let $S_{\max}$ be an upper bound on the number of sprints that may be used to complete the MVP. Following the weighted-sum approach commonly used in multi-objective optimization and project scheduling (Deb 2014, Bomsdorf & Derigs 2008, Liang, Cui, Wang & Demeulemeester 2019), we solve

$$\min \; w_1 \sum_{\sigma=1}^{S_{\max}} y_\sigma + w_2 \sum_{\sigma=1}^{S_{\max}} \sum_{r=1}^{R} \sum_{k=1}^{K_r} c_{rk} q_{\sigma rk}. \tag{1}$$

Table 1. Core notation used in the two-stage model.

Symbol	Meaning
$I = M \cup N$	set of all issues, partitioned into must-have issues M and nice-to-have issues N
sp_i	story points of issue i
P_i	set of predecessors of issue i
req_i	required resource type of issue i
v_i	value obtained if nice-to-have issue $i \in N$ is completed
cap_{rk}, c_{rk}, u_{rk}	capacity, cost, and availability of instance k of resource type r
$x_{\sigma irk}$	fraction of issue i processed in sprint σ by instance k of resource type r
y_σ	binary variable equal to 1 if sprint σ is executed
$q_{\sigma rk}$	number of units of instance k of resource type r used in sprint σ
S^*	number of sprints used to complete the MVP in Stage 1
$N(s, a)$	visit count of state-action pair (s, a) in the RL procedure

The constraints enforce full completion of every $i \in M$, compatibility with the required resource type, capacity limits $\sum_i sp_i x_{\sigma irk} \le cap_{rk} q_{\sigma rk}$, work only in executed sprints ($x_{\sigma irk} \le y_\sigma$), unique start and finish indicators, precedence $\sum_{\varsigma=1}^{\sigma} finish_{\varsigma j} \ge start_{\sigma i}$ for every $j \in P_i$, resource availability $q_{\sigma rk} \le u_{rk} y_\sigma$, and contiguous sprint use $y_\sigma \le y_{\sigma-1}$ for $\sigma \ge 2$.

Stage 2: value maximization. Let S^* be the number of sprints used in Stage 1. Stage 2 starts at sprint $S^* + 1$ and solves

$$\max \sum_{i \in N} v_i \left(\sum_{\sigma=S^*+1}^{D} finish_{\sigma i} \right) - w_3 \sum_{\sigma=S^*+1}^{D} y_\sigma, \tag{2}$$

where w_3 is a small tie-breaking penalty favoring compact use of the remaining sprints. The linking budget constraint is

$$\sum_{\sigma=S^*+1}^{D} \sum_{r=1}^{R} \sum_{k=1}^{K_r} c_{rk} q_{\sigma rk} \le B - \sum_{\sigma=1}^{S^*} \sum_{r=1}^{R} \sum_{k=1}^{K_r} c_{rk} q^*_{\sigma rk}. \tag{3}$$

The remaining Stage 2 constraints mirror those of Stage 1 for resource compatibility, capacity, start/finish logic, precedence, and sprint contiguity.

3 Reinforcement Learning Approach

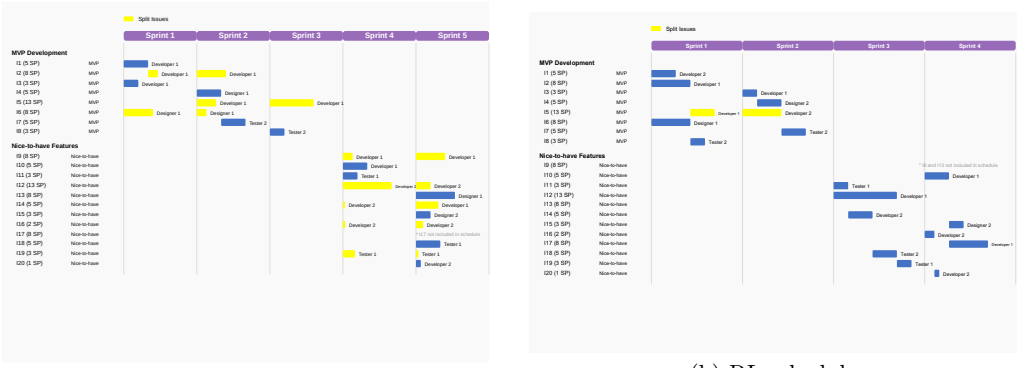
To complement the exact model, we use a two-stage Monte Carlo Control approach. The state s is the current sprint number; all other scheduling information is maintained through auxiliary sets of completed issues (*done*), in-progress issues (*doing*), not-yet-started eligible issues (*to_ do*), and remaining resource capacities. An action a is the selection of one feasible issue to start in the current sprint.

In Stage 1, the per-sprint reward is $R_\sigma = -w_1 \sigma - w_2 c_\sigma$, where c_σ is the sprint cost. In Stage 2, the episode return is the total value of the completed nice-to-have issues, $R_{\text{total}} = \sum_{i \in \text{done}} v_i$. We employ an ϵ-greedy policy and update action values by

$$Q(s, a) \leftarrow Q(s, a) + \frac{R - Q(s, a)}{N(s, a)}, \tag{4}$$

Table 2. Summary of the illustrative-instance outcomes.

Metric	MILP	RL
Total value points	225	200
Total cost	$24,600	$24,600
Schedule length (sprints)	5	4
Completed nice-to-have issues	11	10
Omitted nice-to-have issues	I_{17}	I_9, I_{13}



(a) MILP schedule

(b) RL schedule

Fig. 1. Illustrative comparison of the schedules produced by the MILP model and the RL approach. The MILP solution attains a higher objective value but generates a more fragmented schedule with several split issues across sprints, whereas the RL policy produces a more balanced sprint structure with fewer fragmented tasks.

where $N(s, a)$ is the visit count, i.e., the number of times the state-action pair (s, a) has been observed. Feasibility is enforced by a greedy scheduling routine that assigns available capacity to the cheapest compatible resource instance and carries partially completed issues to later sprints.

4 Illustrative Example and Discussion

We illustrate the models on a hypothetical software project with 20 user stories: 8 MVP issues and 12 nice-to-have issues. Story points follow the Fibonacci scale $\{1, 2, 3, 5, 8, 13\}$, nice-to-have issues are assigned value points between 5 and 50, and the team has three resource types—developer, designer, and tester—with two heterogeneous instances of each type. The total budget is $25,000 and the deadline is five sprints.

Table 2 reports the main quantitative outcomes, while Figure 1 highlights the structural differences between the two schedules. The MILP solution attains 225 value points at a total cost of $24,600, whereas the RL policy attains 200 value points at the same cost. The MILP solution yields the higher objective value, but it also creates a more fragmented plan by splitting some issues across multiple sprints and concentrating many concurrent issues in the same sprint. The RL solution is less fragmented and distributes work more evenly across sprints. We therefore regard the example as an illustration of a practical trade-off between mathematical optimality and execution-friendly sprint structure, rather than as a full computational validation.

Overall, the paper contributes a two-stage agile sprint planning formulation that explicitly separates early MVP delivery from later value maximization under budget, precedence,

resource, and deadline constraints. The illustrative example shows that both approaches can generate feasible sprint plans, and it suggests a trade-off between value optimality and fragmentation. A broader computational study is left for future work.

References

Blazewicz, J., Lenstra, J. K. & Rinnooy Kan, A. H. G. (1983), "Scheduling subject to resource constraints: classification and complexity", *Discrete Applied Mathematics*, 5(1), pp. 11–24. doi:10.1016/0166-218X(83)90012-4.

Bomsdorf, F. & Derigs, U. (2008), "A model, heuristic procedure and decision support system for solving the movie shoot scheduling problem", *OR Spectrum*, 30(4), pp. 751–772. doi:10.1007/s00291-007-0103-6.

Deb, K. (2014), "Multi-objective optimization", in Burke, E. K. & Kendall, G. (eds), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, 2nd edn, Springer, New York, NY, pp. 403–449. doi:10.1007/978-1-4614-6940-7_15.

Karp, R. M. (1972), "Reducibility among Combinatorial Problems", in Miller, R. E. & Thatcher, J. W. (eds), *Complexity of Computer Computations*, Springer, Boston, MA, pp. 85–103. doi:10.1007/978-1-4684-2001-2_9.

Liang, Y., Cui, N., Wang, T. & Demeulemeester, E. (2019), "Robust resource-constrained max-NPV project scheduling with stochastic activity duration", *OR Spectrum*, 41(1), pp. 219–254. doi:10.1007/s00291-018-0533-3.

Best Student Paper Award #1

Constraint and Integer Programming for Resource-Constrained Project Scheduling Problem with Transfer Times

Vilém Heinz[1,2], Zdeněk Hanzálek[2], Christian Artigues[3] and Emmanuel Hébrard[3]

[1] Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic
heinzvil@fel.cvut.cz
[2] Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, Czech Republic
Zdenek.Hanzalek@cvut.cz
[3] LAAS-CNRS, Université de Toulouse, CNRS, France
artigues@laas.fr, hebrard@laas.fr

Keywords: Constraint Programming, Scheduling, RCPSP, Transfer times, Integer Linear Programming.

1 Introduction

The resource-constrained project scheduling problem (RCPSP) is one of the most challenging combinatorial optimization problem as it has both many practical applications and presents a real computational challenge. The RCPSP has many variants and extensions to further increase its applicability. Among the relevant extensions, this paper considers resource setup times also called transfer times. Indeed, these setup times may represent the physical transfer of resource units between an activity A and an activity B. In this case, the transfer time may depend on activity locations, and may not be symmetric: transferring units from A to B may take a longer time than transferring the same units for B to A. Also, some additional resources, e.g. vehicles, may be needed to transfer these units. In this paper, we aim at solving an RCPSP extension that deals with these characteristics of the transfer times. We propose constraint and integer linear programming approaches.

2 Problem definition and related work

Recall the standard RCPSP model with a set of activities $\mathcal{A}$, a set of resources $\mathcal{R}$ and a set of precedence constraints $\mathcal{E}$. Each resource $k \in \mathcal{R}$ has a number of units (capacity) denoted B_k. Each activity $i \in \mathcal{A}$ has a duration $p_i \in \mathbb{N}$ and requires $b_{i,k} \in \mathbb{N}$ units on each resource $k \in \mathcal{R}$. Let $\mathcal{A} = \{1, \ldots, n\}$ and $\mathcal{A}^+ = \mathcal{A} \cup \{0, n+1\}$ where 0 and $n+1$ are dummy activities with $p_0 = p_{n+1} = 0$ and such that for each $i \in \mathcal{A}$, arcs $(0, i)$ and $(i, n+1)$ are present in $\mathcal{E}$. Given a time horizon $\mathcal{T}$, the standard RCPSP formulation aims a assigning a start time S_i to each activity $i \in \mathcal{A}$ and is as follows:

$$\min \quad S_{n+1} \tag{1}$$

$$S_0 = 0 \tag{2}$$

$$S_j \geq S_i + p_i \qquad \forall (i,j) \in \mathcal{E} \tag{3}$$

$$\textstyle\sum_{i \in \mathcal{A}, S_i \leq t \leq S_i + p_i - 1} b_{ik} \leq B_k \; \forall t \in \mathcal{T}, \forall k \in \mathcal{R} \tag{4}$$

The resource units are not explicitly assigned to the activities and the capacity check is solely based on the fact that the resource capacity is not exceeded at each time point. Hence, to model a resource transfer, a resource flow variable $f_{ijk} \geq 0$ must be introduced

to denote the number of units of resource k that must be transferred from i to j after the end of i and before the beginning of j. The time needed for the transfer is denoted $s_{ijk} \geq 0$. In addition, we consider that the transfer may require additional secondary resources gathered in a set $\mathcal{R}'$. Accordingly, resources in $\mathcal{R}$ are called primary resources. Each secondary resource $k' \in \mathcal{R}'$ has a limited capacity $B'_{k'}$. If a transfer actually occurs between two activities i and j on a primary resource (meaning that $f_{ijk} > 0$), the number of units of each secondary resource $k' \in \mathcal{R}'$ is proportional to f_{ijk} according to a coefficient $c_{kk'}$. The time required by resource k' to support the transfer of some primary resource from i to j is denoted $s'_{ijk'} \geq 0$. The presence of both s and s' in the model allows to represent various situations. There are constraints that the transfer activity occupies resource k' non preemptively, lasts $s'_{ijk'}$ units and has to occur inside time window $[S_i + p_i, S_j]$. Once the transfer is completed, the secondary resource units are assumed to be instantaneously available for any other transfer. This means that in case the secondary resource r' is a vehicle that physically transport the resource r from the location of i to the location of j, the empty moves of the vehicle have a negligible time. Given this informal explanation, the problem can now be formally described as follows, where $S_{ijkk'}$ ($C_{ijkk'}$) denote the start (end) of the transfer of the primary resource k units from i to j supported by secondary resource k'.

$$\min \quad S_{n+1} \tag{5}$$

$$S_0 = 0 \tag{6}$$

$$S_j \geq S_i + p_i \qquad \forall (i,j) \in \mathcal{E} \tag{7}$$

$$\textstyle\sum_{i \in \mathcal{A} \cup \{n+1\}} f_{0ik} = B_k \qquad \forall k \in \mathcal{R} \tag{8}$$

$$\textstyle\sum_{i \in \mathcal{A} \cup \{0\}} f_{i,n+1,k} = B_k \qquad \forall k \in \mathcal{R} \tag{9}$$

$$\textstyle\sum_{j \in \mathcal{A} \cup \{0\}} f_{jik} = b_{i,k} \qquad \forall i \in \mathcal{A} \tag{10}$$

$$\textstyle\sum_{j \in \mathcal{A} \cup \{n+1\}} f_{ijk} = b_{i,k} \qquad \forall i \in \mathcal{A} \tag{11}$$

$$f_{ijk} > 0 \Rightarrow S_j \geq S_i + p_i + s_{ijk} \qquad \forall i,j \in \mathcal{A}, \forall k \in \mathcal{R} \tag{12}$$

$$f_{ijk} > 0 \Rightarrow S_{ijkk'} = C_{ijkk'} + s'_{ijk'} \qquad \forall i,j \in \mathcal{A}, \forall k \in \mathcal{R}, \forall k' \in \mathcal{R}', c_{kk'} > 0 \tag{13}$$

$$S_{ijkk'} \geq S_i + p_i \qquad \forall i,j \in \mathcal{A}, \forall k \in \mathcal{R}, \forall k' \in \mathcal{R}', c_{kk'} > 0 \tag{14}$$

$$C_{ijkk'} \leq S_j \qquad \forall i,j \in \mathcal{A}, \forall k \in \mathcal{R}, \forall k' \in \mathcal{R}', c_{kk'} > 0 \tag{15}$$

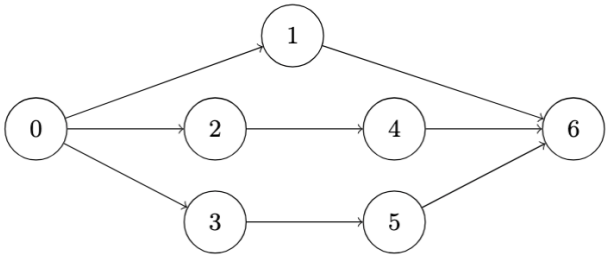
$$\textstyle\sum_{i,j \in \mathcal{A}, k \in \mathcal{R}, S_{ijkk'} \leq t < C_{ijkk'}} c_{kk'} f_{ijk} \leq B'_{k'} \quad \forall t \in \mathcal{T}, \forall k' \in \mathcal{R}' \tag{16}$$

From Objective (5) to Constraints (12), the model is the resource flow model proposed by (Artigues *et al.* 2003) for the RCPSP, with the difference that setup times are included in Constraints (12). Constraints (13) "create" the transfer activity of f_{ijk} units from i to j supported by secondary resource k' (whenever $f_{ijk} > 0$) by setting its completion time to its start time plus its duration.With Constraints (14) and (15), this forces the transfer activity to be performed after the end of i and before the start of j. Finally, Constraints (16) state that each such transfer activity uses $c_{kk'} f_{ijk}$ units of secondary resource k' to support the transfer and ensure that the capacity of k' is not exceeded at any time period.

Such models involving setup/transfer times and secondary resources to support the transfer have already been considered in the literature for generalized shop scheduling problems (Artigues and Roubellat 2001) and for the RCPSP (Krüger and Scholl 2010, Poppenborg and Knust 2016), mostly with heuristics. Exact methods proposed so far are unable to solve instances of more than 10 activities.

3 Illustrative example

Consider an instance with $|\mathcal{A}| = 7,\ |\mathcal{R}| = 2,\ |\mathcal{R}'| = 1,\ B_0 = 4,\ B_1 = 3,\ B'_0 = 3,\ c_{00} = c_{01} = 1$ and the other parameters provided in Fig.1. A solution is displayed in Fig. 2 where arrows represent resource transfers (the plain arrow has length s_{ijk} while the dotted arrow has length $s'_{ijk'}$). The reader can check that when a secondary resource is occupied, the transfer is delayed, such as for the transfer of 2 units of resource 1 from activity 2 to activity 4 that cannot start before time 6 because secondary resource 0 is occupied.



Activity precedence graph.

$\mathcal{A}_i$	p_i	$b_{i,0}$	$b_{i,1}$
0	0	0	0
1	2	3	1
2	2	0	3
3	3	3	0
4	1	0	2
5	4	1	0
6	0	0	0

Activity parameters.

$\mathcal{A}_i$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	-	2	2	1	2	0
2	0	3	-	1	2	2	0
3	0	3	4	-	3	3	0
4	0	1	2	2	-	1	0
5	0	1	2	1	2	-	0
6	0	0	0	0	0	0	0

s_{ij0}

$\mathcal{A}_i$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	-	2	2	2	2	0
2	0	3	-	1	3	1	0
3	0	2	4	-	3	4	0
4	0	1	2	2	-	1	0
5	0	2	2	1	2	-	0
6	0	0	0	0	0	0	0

s_{ij1}

$\mathcal{A}_i$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	-	2	2	1	2	0
2	0	1	-	1	2	1	0
3	0	3	2	-	2	2	0
4	0	1	2	1	-	1	0
5	0	1	1	1	2	-	0
6	0	0	0	0	0	0	0

s'_{ij0}

Fig. 1: Instance parameter definition.

4 Proposed methods and results

We first propose an integer linear formulations to solve the problem, extending the resource flow MILP formulation (Artigues *et al.* 2003). We use a sequencing variable $x_{ijk} \in \{0, 1\}$ equal to 1 when $f_{ijk} > 0$, which allows to linearize implication constraints (12) and (13) with the standard big$-M$ formulation. To linearize secondary resource constraints (16) in the same way, we introduce secondary resource flow variables $\Phi_{i,j,k,a,b,l,k'}$ representing the number of units of secondary resource k' transferred from transfer activity (i, j, k) to transfer activity (a, b, l), as well as sequencing variable $y_{i,j,k,a,b,l,k'}$ In Fig. 2, $\Phi_{2,1,1,3,1,0,0} = 1$ and $\Phi_{3,1,0,2,4,1,0} = 2$ while $y_{2,1,1,3,1,0,0} = y_{3,1,0,2,4,1,0} = 1$.

Unfortunately, this MILP formulation has a huge number of binary variables due to the considerations of all possible transfers. Therefore, we propose a constraint programming (CP) formulation based on the concept of interval variables used to model activities in several solvers such as CPOptimizer, CP-SAT, OptalCP, Hexaly.... We introduce an interval variable for each activity $i \in \mathcal{A}$ and an optional interval variable for each possible transfer (i, j, k, k').

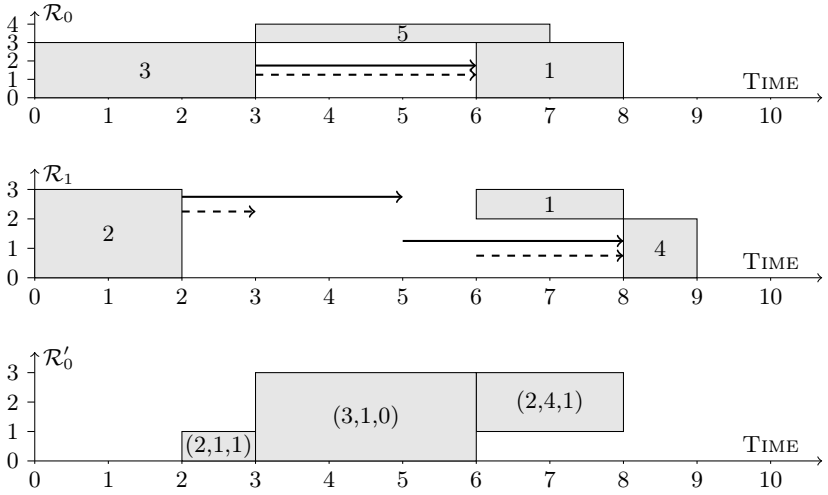


Fig. 2: Solution of the illustrative example.

We also propose a greedy randomized heuristic (GRH) to obtain a solution to the global problem and that can be used as a warm-start solution to the CP model.

Finally, we propose a multi-phase heuristic based on CP and ILP. We first solve by CP the standard RCPSP ignoring all transfers. we store the pairwise time distance τ_{ij} between activities in the solution, the precedence constraints $\mathcal{E}'$ induced by the solution and the transfers $\mathcal{I}$ inconsistent with the solution. (e.g. if i precedes j in the solution then the transfer from j to i is forbidden). By searching a feasible primary resource flow assignment given $\mathcal{E}'$ and $\mathcal{I}$, the search space is highly reduced and we aim at creating arcs that preserve the $\tau_{i,j}$ (i.e. such that $s_{i,j,k} \leq \tau_{i,j}$). A right-shifting heuristic then obtains a global feasible solution (i.e. also satisfying the secondary resource constraints) by keeping the primary flows obtained at the previous and the partial order from the first step. Finally, the CP method is warm-started with the so-obtained upper bound.

The table below reports the average gap of the CP model and the multi-phase method from the best known lower bound on a set of instances from 30 to 120 activities. The results show that the multi-phase method outperforms the CP model.

Approach	j30	j60	j90	j120
CP model warm started by GRH	0.13	0.20	0.23	0.23
Multi-phase method	0.00	0.00	0.00	0.00

Acknowledgments
This work was funded by the Grant Agency of the Czech Republic under Project GACR 25-17904S.

References

C. Artigues, P. Michelon and S. Reusser, 2003, "Insertion techniques for static and dynamic resource-constrained project scheduling", *European Journal of Operational Research*, vol. 149(2), pp. 249–267.

C Artigues and F Roubellat, 2001. "A Petri net model and a general method for on and off-line multi-resource shop floor scheduling with setup times", *International Journal of Production Economics* vol. 74 (1-3), pp 63–75

J. Poppenborg and S. Knust, 2016 "A flow-based tabu search algorithm for the RCPSP with transfer times". *OR Spectrum*, vol. 38(2), pp. 205-344

D. Krüger and E. Scholl, 2010. "Managing and modelling general resource transfers in (multi-) project scheduling". *OR Spectrum*, vol. 32(2), pp 369–394.

Learning to Schedule the Final Assembly Line: a GNN for Fast, Stable Scheduling under Uncertainty

Sergei Gladyshev[1], Olga Battaia[1], Romain Guillaume[2], Philippe Ruiz[1]

[1] KEDGE Business School, Bordeaux, France
{sergei.gladyshev02, olga.battaia, philippe.ruiz}@kedgebs.com
[2] IRIT — Universite de Toulouse, Toulouse, France
romain.guillaume@irit.fr

Keywords: aircraft final assembly line, project scheduling, stability, deep neural networks, uncertainty.

1 Introduction

Scheduling tasks in aircraft Final Assembly Lines (FALs) is a challenging combinatorial problem: hundreds of jobs must be allocated to a limited pool of workers under precedence and resource constraints. In practice, processing times are uncertain (variable working conditions, possible delays), and maintaining schedule stability is crucial because many logistic activities depend on initial task start times. Frequent changes, therefore, incur high operational costs.

Classic optimization approaches for scheduling, such as Mixed Integer Linear Programming (MILP) and Constraint Programming (CP), can produce high-quality schedules for deterministic formulation of the problem but typically require substantial computation time to take into account uncertain processing times. We propose a learning-based alternative: a Graph Neural Network (GNN) that, given the current partial schedule and job duration distributions, quickly predicts which jobs should be scheduled next, by which workers, and how much buffer time to insert. We train the model by imitating robust schedules produced by a CP solver using Sample Average Approximation (SAA) with explicit buffer allocation, and we evaluate both the fidelity of the predictions to CP solutions and the final schedule quality obtained by iteratively applying the GNN. Results on synthetic FAL instances (30, 60, 120 jobs) show that the GNN yields competitive schedules with greatly reduced runtime.

2 Problem formulation

We consider a single assembly station modeled as a Stochastic Resource-Constrained Project Scheduling Problem (SRCPSP). Let $J = \{1, \dots, n\}$ be the set of jobs and $W = \{1, \dots, m\}$ the pool of identical workers. Each job $j \in J$ requires one worker and has a stochastic processing time P_j with known distribution and expected value $\mathbb{E}[P_j]$. Precedence constraints are represented by a partial order: $i \prec j$ means job j cannot start before job i completes. A worker can process at most one job at a time (capacity constraint).

Even if the duration of jobs is uncertain, an **initial schedule**, for instance, considers initial durations as $p_j^{init} = \mathbb{E}[P_j]$. When the initial plan is executed, the actual durations may differ from the planned durations, and the schedule has to be adjusted. Formally, let s_j^{init} denote the start time in the initial schedule and s_j the start time in a **realized plan**. Since many secondary processes depend on the initial schedule, we restrict $s_j \geq s_j^{init} \ \ \forall j$. Also, in the realized schedule, job assignment and execution order should be identical to the initial plan. We treat start-time delays in the realized schedule relative to that initial

plan as costs. We adopted the sum of expected start-time deviations

$$\text{Stability} = \mathbb{E}\left[\sum_{j \in J}(s_j - s_j^{init})\right]$$

as a measure of schedule stability. Makespan and stability are objective functions. The deterministic version of this problem with makespan as objective function is known to be NP-hard (Garey, M.R. and Johnson, D.S. 1978).

3 Baseline CP model and training data generation

We generate training labels using a CP-based approach that combines Sample Average Approximation (SAA) and explicit buffer-time allocation. Specifically:

- We sample N scenarios of job durations. Scenario 1 uses $p_j^{init} = \mathbb{E}[P_j]$; the remaining $N-1$ scenarios sample durations from the distributions of P_j
- The CP solver is given the N scenarios and allowed to choose, for any "*bound*" jobs in scenario 1, increased (alternative) durations. Where "*bound*" is a fixed parameter in advance. This effectively reserves buffer time after chosen jobs.
- Constraints enforce that no job may start earlier than in scenario 1, and jobs cannot be reassigned to different workers (to preserve the initial assignment).
- The objective function minimizes the makespan and stability function in lexicographical order.

We implement the model in IBM ILOG CP Optimizer 22.1.1, following the pseudo-code in Algorithm 1.

Parameters: The model sets the number of scenarios N, the duration matrix P (lines 1–2), and binary buffer times (lines 3–4). The parameter `bound` fixes how many jobs may receive buffer = 1. **Variables:** Interval variables for each job and scenario are created in lines 5 and 7. In scenario 1, optional alternative intervals encode buffer choices (line 6). Optional worker-assignment variables $x_{j,k}^s$ (line 8) and sequence variables seq_k^s (line 9) model the assignment and execution order of jobs. **Constraints:** Precedence constraints appear in line 10. Worker choice and buffer-choice alternatives are set in lines 11 and 15. No-overlap constraints ensure resource capacity (line 12). Same-sequence constraints (line 13) enforce identical assignments and execution orders across scenarios. Start-before-start constraints (line 14) prevent earlier starts in scenarios $2, \ldots, N$ than in scenario 1. A cardinality constraint fixes the number of nonzero buffers (line 16). **Objective:** The lexicographic objective (lines 17–18) minimizes (i) makespan in scenario 1 and then (ii) the stability metric across scenarios.

The CP output is a stable schedule across scenarios; we extract from each solved instance a sequence of (partial schedule; next-dispatched-job(s), assigned worker, buffer before job) pairs to construct supervised training data. This process produces a dataset of partial schedules and the corresponding "next actions" recommended by CP under uncertainty and stability constraints. These are the targets used to train the GNN.

4 GNN model

Graph Neural Networks are gaining extreme popularity for job shop scheduling problems (Smit, I.G. *et. al.* 2025). We applied a similar approach to our specific stochastic RCPSP. We encode each instance as a directed acyclic graph (DAG) of precedence relations and then enrich it to reflect the current partial schedule (see example in Fig. 1):

Algorithm 1 CP baseline model

1: $N \leftarrow ScenariosNum$ ▷ Set the number of considering scenarios
2: $P \leftarrow [[p_j^s \mid j = 1 \dots n] \mid s = 1 \dots N]$ ▷ The matrix of the jobs' duration in the all scenarios
3: $\vec{b_j} \leftarrow [0, 1] \ \ \forall j = 1 \dots n$ ▷ Possible buffer times for the first scenario
4: $bound \leftarrow TotalBufTime$ ▷ The sum of buffer times
5: $x_j^1 \leftarrow intervalVar() \ \ \forall j = 1 \dots n$ ▷ Jobs in the first scenario
6: $x_j^{1,r} \leftarrow intervalVar(size = p_j^1 + \vec{b_j}[r], \textbf{optional}) \ \ \forall j = 1, \dots, n; \ r = 0, 1$
7: $x_j^s \leftarrow intervalVar(size = p_j^s) \quad \forall s = 2 \dots N; \ j = 1 \dots n$ ▷ Jobs in the other scenarios
8: $x_{jk}^s \leftarrow intervalVar(\textbf{optional}) \ \ \forall s = 1 \dots N; \ j = 1 \dots n; \ k = 1 \dots m$ ▷ Assignment variables
9: $seq_k^s \leftarrow sequenceVar([x_{jk}^s \mid j = 1 \dots n]) \ \ \forall s = 1 \dots N; \ k = 1 \dots m$ ▷ The job sequences
10: $endBeforeStart(x_i^s, x_j^s) \ \ \forall s = 1 \dots N; \ (i \prec j)$ ▷ Precedence constraints
11: $alternative(x_j^s, [x_{jk}^s \mid k = 1 \dots m]) \ \ \forall s = 1 \dots N; \ j = 1 \dots n$ ▷ Assignment constraints
12: $noOverlap(seq_k^s) \ \ \forall s = 1 \dots N; \ k = 1 \dots m$ ▷ No overlap constraints
13: $sameSequence(seq_k^1, seq_k^s) \ \ \forall s = 2 \dots N; \ k = 1 \dots m$ ▷ The job sequences are equal
14: $startBeforeStart(x_j^1, x_j^s) \ \ \forall s = 2 \dots N; \ j = 1 \dots n$ ▷ Right-shift policy
15: $alternative(x_j^1, [x_j^{1,r} \mid r = 0, \dots, l]) \ \ \forall j = 1, \dots, n$ ▷ Choose buffer time
16: $\sum_j \sum_r \vec{b_j}[r] \cdot presenceOf(x_j^{1,r}) = bound$ ▷ Limit total buffer time
17: **minimizeStaticLex** $\{endOf(x_{max}^1); \bigoplus_{s=1}^N F(s)\}$ ▷ $endOf(x_{max}^1)$ is an initial makespan
18: $\bigoplus_{s=1}^N F(s) = \sum_{s=2}^N \sum_{j=1}^n (startOf(x_j^s) - startOf(x_j^1))$ ▷ Minimize the Stability

– add execution-sequence arcs for already-scheduled jobs;
– append, for each worker, a virtual node representing the end of its current sequence (handles idle workers);
– mark as *candidates* the jobs whose predecessors are all scheduled (eligible next jobs);
– add reverse precedence and reverse-sequence arcs to improve message flow and convergence.

Node features comprise min, max, $\mathbb{E}[P_j]$, a scheduled flag, start time, worker ID, completion time distribution parameters (when scheduled), a virtual-node flag, and a candidate flag (see example in Fig. 1c). Edge features indicate edge types (precedence, execution-sequence, reverse).

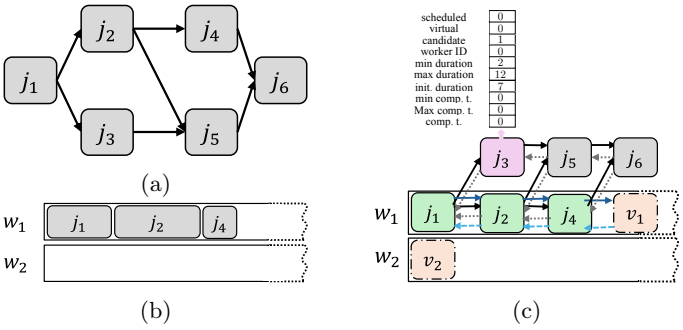


Fig. 1: Example of transforming precedence constraints (a) and a partial schedule (b) into a GNN graph (c) for 6 jobs $j_1, \dots, j_6$ and 2 workers w_1, w_2. Jobs j_1, j_2, and j_4 are already assigned to w_1. In (c), orange nodes v_1 and v_2 represent virtual end of sequence jobs; j_3 is a single candidate.

A message-passing GNN computes node embeddings with L rounds of convolution over all edge types using `GATv2Conv` (PyTorch), hidden dimension `hid_dim`, ReLU activations, and a final projection to `out_dim`. The model has two prediction heads:

1. a perceptron classifier on each *candidate* node that outputs the probability the job requires a buffer before execution (binary classifier);
2. a perceptron link-classifier for each (virtual-node, candidate) pair that takes the concatenation of their output embeddings and predicts a score for assigning that candidate to the worker represented by the virtual node (link prediction/assignment).

Training minimizes binary cross-entropy (BCEWithLogits) for both buffer and link predictions. At inference, we iteratively select the highest-scoring (virtual, candidate) links to fix assignments and buffer decisions.

5 Experiments and results

We generate 600 random instances with $n \in \{30, 60, 120\}$ jobs and 5 workers. Each job's processing time is a discrete uniform distribution with randomly selected parameters $P_j \in \mathcal{U}\{p_j^{init} - l_j; p_j^{init} + l_j\}$, where p_j^{init} and l_j are generated from $\mathcal{U}\{5; 10\}$ and $\mathcal{U}\{1; 3\}$, respectively. For each instance, we solve the CP model to obtain target schedules and construct the supervised dataset of (partial schedule, next jobs) pairs.

After shuffling, we split the dataset into training, validation, and test sets in a 0.8/0.1/0.1 ratio. We evaluate imitation accuracy (ROC AUC) and runtime. On the test instances, the GNN achieves solid imitation performance (ROC AUC ≥ 0.75) for both worker assignment of candidate jobs and buffer-time prediction.

6 Conclusion and future work

We presented a GNN-based scheduler that rapidly predicts next dispatch decisions for a single-station FAL with stochastic processing times and precedence constraints. By imitating CP solutions generated with SAA and explicit buffers, the model learns to trade off schedule quality and stability while enabling fast rescheduling. So far, we have tested the model that predicts only the next job to schedule. Future work will investigate (i) using the model for generating a complete schedule in a beam search or stochastic sampling to mitigate rollout error (ii) integrating the GNN into a Deep Reinforcement Learning (DRL) framework.

Acknowledgements

Our work has benefited from the AI Interdisciplinary Institute ANITI. ANITI is funded by the French "Investing for the Future – PIA3" program under the Grant agreement n°ANR-23-IACL-0002.

References

CPOptimizer (2022) *IBM ILOG CP Optimizer User's Manual*. IBM. Available at: `https://www.ibm.com/docs/ru/icos/22.1.1?topic=optimizer-cp-users-manual`.

Garey, M.R. and Johnson, D.S., 1978, "Strong" NP-completeness results: Motivation, examples, and implications, *Journal of the ACM*, Vol. 25(3), pp. 499-508.

Smit, I.G., Zhou, J., Reijnen, R., Wu, Y., Chen, J., Zhang, C., Bukhsh, Z., Zhang, Y. and Nuijten, W. (2025) Graph neural networks for job shop scheduling problems: A survey, *Computers & Operations Research*, 176, p. 106914.

Best Student Paper Award #2

Scheduling under multi-skill workforce constraints: A novel Logic-Based Benders Decomposition Approach

Johanna Mlekusch[1],Carla Juvin[2], Christian Artigues[3] and Richard F. Hartl[1]

[1] University of Vienna, Austria
johanna.mlekusch@univie.ac.at, richard.hartl@univie.ac.at
[2] TBS Business School, Toulouse, France
c.juvin@tbs-education.fr
[3] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
christian.artigues@laas.fr

Keywords: Logic-Based Benders Decomposition; Constraint Programming; Dual-Resource Constraints; Multi-Skill Workforce; Re-entrant Flow Shop Scheduling

1 Introduction

This research is motivated by a real-world production planning problem encountered in the screen printing industry. The production system is modeled as a Dual-Resource-Constrained Re-entrant Flexible Flow Shop Problem (DRCRFFSP), which is a variant of the classical Re-entrant Flow Shop Problem (RFSP). Unlike to the standard Flow Shop Problem (FSP), where jobs progress through all stages in a fixed sequence once, the re-entrant characteristic considered here allows jobs to revisit specific stages multiple times before completion. Additionally, the problem involves a heterogeneous, multi-skilled workforce and requires skilled workers to operate the machines throughout the entire processing time. As a result, the schedule must not only adhere to precedence constraints but also account for resource limitations while ensuring that tasks are assigned to workers with the necessary skills.

Given that the FSP with n jobs and more than two machines is NP-hard (Pinedo 2008), designing efficient solution methods for such an extended setting is particularly challenging. Prior work introduced a constraint programming (CP) formulation and a hybrid genetic algorithm (Mlekusch and Hartl 2024), which were able to generate feasible solutions quickly for large-scale instances, although with limited ability to prove optimality.

To address these limitations, the present research introduces an exact solution method based on a novel Logic-Based Benders Decomposition (LBBD) scheme. In most existing benders decomposition approaches for scheduling problems involving resource flexibility, the assignment problem usually acts as the master problem (Juvin *et al.* 2023). In contrast, this methodology uses a decomposition in which the scheduling problem acts as the master problem. It fixes the timing of operations, while a graph colouring subproblem with forbidden colours handles the assignment of operations under skill constraints and identifies combinations of operations that cannot be assigned.

Computational results demonstrate that the proposed exact method outperforms an integrated CP approach in terms of the number of instances solved to optimality and the strength of the bounds obtained for instances that remain unsolved within the time limit. Overall, these results show that the proposed LBBD scheme is an effective exact solution method for the DRCRFFSP and may also be suitable for other complex scheduling problems with limited multi-skilled resources.

2 Problem description

In the DRCRFFSP, a set of jobs $J = \{1, \ldots, n\}$ is processed on stages $S = \{1, \ldots, g\}$ in a fixed order. Each job $j \in J$ consists of operations $O_j = \{1, \ldots, k_j\}$ that may revisit stages, i.e., jobs may traverse the ordered stage sequence multiple times while possibly skipping stages. Although comparable to a Job Shop Problem (JSP) (Pinedo 2008), the job routing is not arbitrary since all jobs follow the same global stage order.

At every stage $s \in S$, there are some identical parallel machines $m \in M_s$ and several workers $w \in W_s$ who can operate on the corresponding machines.

Each worker w is qualified to operate machines on a subset of stages $S_w \subseteq S$, such that $w \in W_s$ if and only if $s \in S_w$. Each operation o_{ji} requires the simultaneous availability of one machine $m \in M_{s\hat{j}i}$ and one suitably skilled worker $w \in W_{s\hat{j}i}$ at its respective stage $s\hat{j}i$ throughout the processing time p_{ji}. Processing times are deterministic and independent of the assigned worker and machine. The objective is to minimize the maximum completion time. Figure 1 depicts an example of a schedule with 3 jobs, 3 stages (with 2-2-1 machines, respectively) and 3 workers.

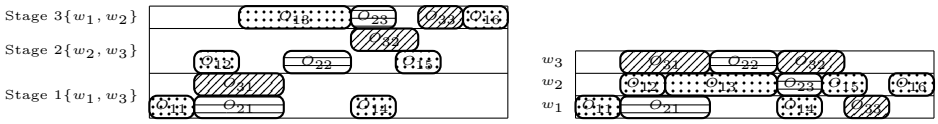


Fig. 1: Example of the DRCRFFSP

3 Logic-based Benders Decomposition

The solution method, first introduced by (Hooker and Yan 1995), builds upon the classic Benders decomposition, which typically separates optimization problems into a master problem and one or more subproblems.

The DRCRFFSP can be decomposed into a scheduling master problem and an assignment subproblem. The master problem represents the classical RFFSP, which can be efficiently solved using CP methods (Mlekusch and Hartl 2024). To account for the resources of workers already in the master problem, we add a constraint ensuring that the number of operations processed simultaneously does not exceed the number of available workers $|W|$. Figure 2 shows how a part of the solution to the master problem could look for the example instance shown in Figure 1. By solving the master problem, we obtain a lower bound (LB) for the global problem.

Before solving the subproblem, we solve an additional scheduling problem using CP to obtain an upper bound (UB). In this UB-model, we fix the sequence of operations on each machine from the master problem, but not the exact start times, while we determine the worker assignment. Any feasible solution to this model provides a valid UB for the global problem.

The subproblem involves assigning avaialable workers to the scheduled operations and thereby identifies conflicting operations. Each operation o_{ji} can only be performed by a subset of workers, $W_{s\hat{j}i}$, who possess the required skills. Two operations that overlap in time must be assigned to different workers. Hence, the subproblem can be formulated as an interval graph coloring problem with forbidden node colors, which is NP-complete.

Through the subproblem, we identify minimal conflicts, defined as subsets of operations for which no feasible worker assignment exists. Removing any single operation or overlap between operations in such a subset restores feasibility. Figure 2 illustrates an example of such a conflict as operation o_{13} cannot be assigned in the current schedule. To identify these conflicts, we use a complete enumeration method based on the recursive list-coloring algorithm of (Zeitlhofer and Wess 2003), which exploits the interval structure and runs in pseudo-polynomial time. These conflicts are then translated into feasibility cuts and added to the master problem, thereby eliminating infeasible solutions and guiding the search towards an optimal schedule.

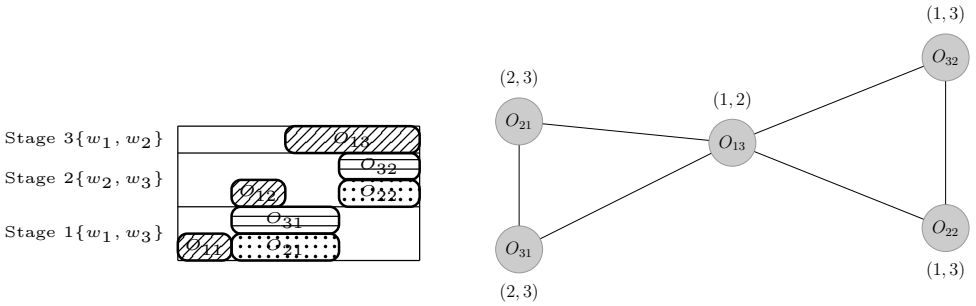


Fig. 2: Example of a conflict, with two cliques

4 Computational study

A preliminary computational study was carried out on a set of instances comprising up to 7 jobs, 5 stages, 4 workers and 2 re-entries. To generate the set of instances for the DRCRFFSP we use the parameter shown in the Table 1.
The distribution of skills among the workers is determined as follows: Each worker has a 50% probability of having skills for a given stage. In cases where a worker has no skills, a random stage is assigned to that worker. For each stage, we randomly select several parallel machines from the range [1,4]. To maintain a balance between the available machines and the number of jobs to be produced, we ensure that the number of machines never exceeds the number of jobs. If the number of workers qualified for a particular stage is less than the number of machines in parallel, a corresponding number of workers is randomly selected for that stage, allowing all machines to run in parallel if necessary. Following this scheme, we generate 3 instances, each characterised by a different skill and machine distribution, for each possible combination of the parameters jobs and stages, resulting in a total of 36 instances.

5 Results

The proposed Logic-Based Benders Decomposition (LBBD) approach was compared with a CP formulation of the integrated problem, implemented using IBM ILOG CP Optimizer. Both methods were executed within a time limit of 1800 seconds using the default solver settings.

A summary of the computational results is reported in Table 2. The LBBD approach demonstrated greater performance than the CP formulation, solving 31 out of 36 instances to proven optimality, versus 23.

Of the 22 instances in which both approaches reached optimality, LBBD achieved an average computation time of 38.22 seconds, which was slightly lower than the CP formulation's average time.

In the four instances where neither approach could prove optimality within the time limit, the LBBD produced higher-quality solutions, with an average gap of 0.68% to the best known solution, compared to 23.25% for the CP formulation. In one instance, the CP formulation was the only approach that could find and prove the optimal solution, taking 675.09 seconds, whereas the solution obtained by LBBD had an optimality gap of 1.59%.

Finally, LBBD uniquely solved 9 instances to optimality with an average runtime of 23.53s. For these instances, the CP formulation failed to prove optimality and reported an average optimality gap of 32.46%.

Overall, these results indicate that the LBBD framework provides superior solution quality compared to the CP formulation for the integrated problem on small DRCRFFSP instances, particularly by providing tighter lower bounds.

Future work will focus on evaluating the approach with larger instances involving more jobs, stages and workers, as well as alternative problem settings, in order to further assess scalability and generalisability.

Parameter for instances	Tiny
Nr. of jobs	$[5, 7]$
Nr. of stages	$[2, 5]$
Nr. of re-entrances	$\{2\}$
Nr. of machines/stage	$[1, 4]$
Processing times	$[1, 10]$
Nr. of workers	$\lceil 0.6m \rceil$
Missing operations	20%

Table 1: Parameters for instance generation

Category	**Nr. Inst.**	**CP**	**LBBD**
Both optimal	22	38.83 s	38.22 s
Only CP optimal	1	675.09 s	1.59%
Only LBBD optimal	9	32.46%	23.53 s
Neither optimal	4	23.25%	0.68%
Total	**36**	23 optimal	31 optimal

Table 2: Comparison of CP and LBBD over 36 instances (time limit: 1800s). *Entries in seconds (s) denote average runtime, while entries in % denote the average optimality gap*

References

Hooker J. N., H. Yan, 1995, "Logic circuit verification by Benders decomposition", *Principles and Practice of Constraint Programming: The Newport Papers*, pp. 267–288, MIT Press, Cambridge, MA.

Juvin C., L. Houssin, P. Lopez, 2023, "Logic-based Benders decomposition for the preemptive flexible job-shop scheduling problem", *Computers & Operations Research*, Vol. 152, pp. 106-156, doi:10.1016/j.cor.2023.106156.

Mlekusch J., R. F. Hartl, 2024, "The Dual-Resource-Constrained Re-entrant Flexible Flow Shop: A Constraint Programming Approach and a Hybrid Genetic Algorithm", *International Journal of Production Research*, Vol. 63(5), pp. 1803-1824.

Pinedo M., 2008, "Scheduling: Theory, Algorithms, and Systems", 3rd ed., Springer, New York, NY, 678 pp., doi:10.1007/978-0-387-78935-4.

Zeitlhofer T., B. Wess, 2003, "List-coloring of interval graphs with application to register assignment for heterogeneous register-set architectures", *Signal Processing*, Vol. 83(7), pp. 1411–1425.

Solution approaches for the multi-interval resource investment problem

Laura Henke[1], Lena Sophie Wohlert[1] and Jürgen Zimmermann[1]

Clausthal University of Technology, Germany
laura.henke@tu-clausthal.de, lena.sophie.wohlert@tu-clausthal.de,
juergen.zimmermann@tu-clausthal.de

Keywords: project scheduling, multi-interval resource investment, serial generation scheme.

1 Introduction

In Germany, energy-intensive companies must pay for their electricity consumption, as well as for their annual or monthly peak power (§17, StromNEV). This peak can be caused either by a single energy-intensive process, or the overlap of several activities. Especially in the latter case, the objective function for a project scheduling problem should be the sum of resource peaks over disjoint time intervals, which must be minimized. To our knowledge, this objective is novel and may also be of interest for other monthly or periodically paid resources like software licences, server capacities, or temporary workers. Therefore, we extend the well-known resource investment problem (RIP), also known as the resource availability cost problem (Kreter *et al.* 2018), to a multi-interval RIP.

2 Problem Formulation and Mathematical Model

A project is decomposed into n real activities as well as dummy activities 0 and $n+1$ for the project start and completion. Each activity $i \in V = \{0, \ldots, n+1\}$ has a start time $S_i \in \mathbb{N}$ and the start times of all the activities create a schedule $\mathcal{S} = (S_0 \ldots, S_{n+1})$. We assume that a project always starts at $S_0 = 0$. The project is modelled as an activity-on-node network $N = (V, E, \delta)$, where each arc $\langle i, j \rangle \in E$ with weight $\delta_{ij} \in \mathbb{Z}$ represents either a minimum or maximum time lag between the start of two activities. The maximum project duration is given by $\bar{d}$ and is met by an arc from the project completion to the start. Let d_{ij} be the length of a longest path in N from activity i to activity j. With these, the earliest start time $ES_i := d_{0i}$ and the latest start time $LS_i := -d_{i0}$ can be determined, resulting in the set of feasible start times $\overline{W}_i$. Each activity i has a processing time $p_i \in \mathbb{N}$, with $p_0 = p_{n+1} = 0$. During the project, a set of renewable resources $\mathcal{R} = \{1, \ldots, R\}$ is used and the amount of resource k required by activity i is given by r_{ik}.

We introduce a partition of the time horizon $[0, \bar{d}]$ into intervals of length Δ, resulting in the set of intervals $\mathcal{T}$, where each interval $\tau \in \mathcal{T}$ spans $[(\tau-1)\cdot\Delta, \min(\tau\cdot\Delta, \bar{d})[$. The cost for resource k in interval τ is denoted as $c_{k\tau}$. The objective is to find a feasible schedule minimizing the sum of cost for the maximum resource consumption over all intervals $\mathcal{T}$. For $\Delta = \bar{d}$ the multi-interval RIP equals the classic RIP. Consequently, the multi-interval RIP is NP-hard like the classic RIP (Kreter *et al.* 2018). The problem is modelled with binary variables x_{it} indicating if activity i starts at time t according to Pritsker *et al.* (1969). Variable $z_{k\tau}$ represents the maximal utilization of resource k in interval τ.

$$
\begin{aligned}
\min \quad & \sum_{k\in\mathcal{R}} \sum_{\tau\in\mathcal{T}} c_{k\tau} \cdot z_{k\tau} & & \\
\text{s.t.} \quad & \sum_{t\in\overline{W}_j} t \cdot x_{jt} - \sum_{t\in\overline{W}_i} t \cdot x_{it} & \geq \delta_{ij} \quad & \forall\langle i,j\rangle \in E \\
& \sum_{t\in\overline{W}_i} x_{it} & = 1 \quad & \forall i \in V \\
& \sum_{i\in V} r_{ik} \sum_{t'=\max(ES_i,t-p_i+1)}^{\min(t,LS_i)} x_{it'} & \leq z_{k\tau} \quad & \forall k \in \mathcal{R}, \forall \tau \in \mathcal{T}, t \in [(\tau-1)\cdot\Delta, \tau\cdot\Delta[\\
& x_{00} & = 0 & \\
& z_{k\tau} & \geq 0 \quad & \forall k \in \mathcal{R}, \forall \tau \in \mathcal{T} \\
& x_{it} & \in \{0,1\} \quad & \forall i \in V, t \in \overline{W}_i
\end{aligned}
$$

3 Structural properties

For the classic RIP, a search space reduction is possible due to structural properties, which we outline before exploring if something similar applies to the multi-interval RIP. A strict order is implied if an activity j starts after the end of activity i, i.e. $S_j \geq S_i + p_i$. All strict orders between pairs of real activities $(i,j) \in \{1,\ldots,n\}^2$ that are satisfied by a schedule S are denoted as the schedule-induced order $O(S)$. Furthermore, the equal order set consists of all schedules that induce the identical strict order and is denoted by $S_T^=(O(S)) := \{S' \in S_T | O(S) = O(S')\}$, where S_T is the set of all time-feasible schedules. The classic RIP-objective is constant on equal order sets and lower semicontinuous. Therefore, it is locally regular and it is sufficient to consider the minimal points of all order polytopes. In such schedules, each activity $i \in V$ either starts at $S_i = ES_i$ or immediately after at least one other activity $j \in V$ i.e. $S_i = S_j + p_j$. These schedules are called quasi-active schedules (Kreter *et al.* 2018). Figure 1 illustrates the structural properties of the multi-interval RIP. In the given example, the start times of activities j and h are fixed, while the start time of activity i remains open, ranging from $ES_i = 0$ to $LS_i = 5$. The cumulative resource consumption, which depends on the schedule S and the point in time t, is denoted as $r(S,t)$ and is depicted in the resource profile on the left. Based on the resource profile, it is evident that three distinct schedule-induced orders arise depending on S_i, as listed on the right. The middle of the figure depicts the objective value $f(S_i)$ for the multi-interval RIP, illustrating that objective function remains lower semicontinuous. Regarding the intervals, the objective value can generally increase if a new interval starts when an activity is processed. Furthermore, the objective value can decrease if an activity starts exactly at an interval start. To reflect this, we can insert for each interval $\tau \in \mathcal{T}$ a fictitious activity with a processing time of zero at its start $(\tau-1)\cdot\Delta$. Considering the extended set of activities, the objective function remains constant on equal order sets and thus locally regular. Therefore, there is always an optimal schedule in which S_i of each activity $i \in V$ satisfies one of the following: $S_i = ES_i$, $S_i \in \{(\tau-1)\cdot\Delta \mid \tau \in \mathcal{T}\}$ or there is an activity $j \in V$ with $S_i = S_j + p_j$.

4 Solution approaches

The reduction of the solution space can be exploited in heuristic approaches. Algorithm 1 is an easy construction heuristic based on a serial generation scheme. If the given project network does not contain a cycle of positive length, the algorithm returns a feasible schedule $\mathcal{S}$. We use a multi-start approach that repeats the algorithm as long as a *timelimit* is not reached. To construct a schedule, we start by fixing the start times of activities i with $ES_i = LS_i$ to $S_i^C := ES_i$. After defining C and $\overline{C}$, an activity $j \in \overline{C}$ is chosen based

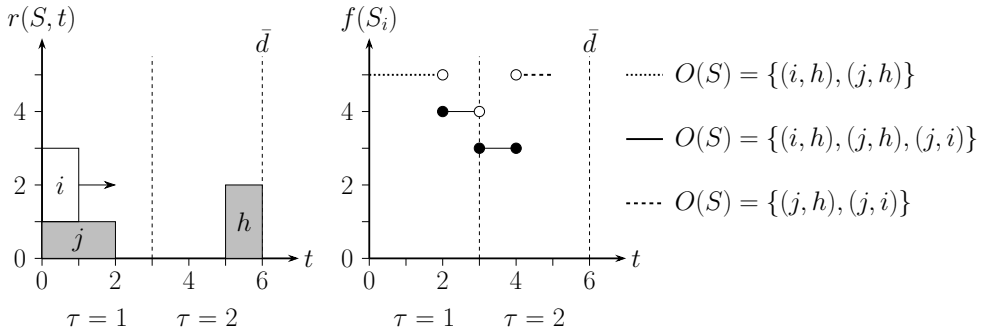


Fig. 1. Example for structural properties

on selection strategy 1. For example, the selection of an activity could be based on a priority list containing all activities $j \in \overline{C}$ in descending order of resource utilization. The decision set D_j of start times S_j is determined according to the structural properties of the problem. For each start time in D_j the expansion cost are calculated. The objective value of the partial schedule S^C with the activities in C is subtracted from the objective value of $S^{C\cup\{j\}}$ with activities in $C \cup \{j\}$. The start time of j is chosen by selection strategy 2, which could be a greedy strategy that chooses t^* with minimal expansion cost or the choice is made randomly. Activity j is postponed from set $\overline{C}$ to C. Then, the earliest and latest start times of the remaining activities in $\overline{C}$ are updated. Once all activities are scheduled, post-processing is performed. The activities are rescheduled in ascending order of activity index. Activity j is removed from the current solution S and D_j is determined. Then, expansion cost are calculated and the start time S_j is set to $t^* \in D_j$ with minimal expansion cost. If the objective value could be improved by changing the start of activity j, the value of S_j is updated in the schedule S and the process is repeated with activity $j+1$.

Algorithm 1 Construction heuristic with different selection strategies

1: Determine d_{ij} $\forall i,j \in V$, and $ES_i := d_{0i}$ and $LS_i := -d_{i0}$ $\forall i \in V$
2: set $C := \{0\}$, $S^C := (0)$ and $\overline{C} := V \setminus \{0\}$
3: **for** $i \in \overline{C}$ with $ES_i = LS_i$ **do** set $S_i^C := ES_i$, $C := C \cup \{i\}$ and $\overline{C} := \overline{C} \setminus \{i\}$
4: choose activity j according to selection strategy 1 and determine $D_j(S^C)$
5: **for** $t \in D_j(S^C)$ **do** calculate expansion cost $f^a(S^C, j, t)$
6: choose start time $t^* \in D_j$ according to selection strategy 2
7: set $S_j^C := t^*$, $C := C \cup \{j\}$ and $\overline{C} := \overline{C} \setminus \{i\}$
8: **for** $i \in \overline{C}$ **do** $ES_i := \max(ES_i, S_j + d_{ji})$ and $LS_i := \min(LS_i, S_j - d_{ij})$
9: **for** $i \in \overline{C}$ with $ES_i = LS_i$ **do** set $S_i^C := ES_i$, $C := C \cup \{i\}$ and $\overline{C} := \overline{C} \setminus \{i\}$
10: do post processing

5 Preliminary results

The experiments were made using the UBO-testsets[1] (Schwindt 1998), which were solved by Gurobi as well as five heuristic variants. Selection strategy 1 to choose an activity

[1] available online: `https://www.wiwi.tu-clausthal.de/abteilungen/betriebswirtschaftslehre-insbesondere-produktion-und-logistik/forschung/schwerpunkte/single-mode-project-duration-problem-rcpsp/max`

j is either `random` or `greedy`. Further, selection strategy 2, which determines the start time t^* of activity j, are `random`, `greedy` or `roulette wheel`. For the latter, selection probabilities were based on expansion cost. The combination of `greedy` and `greedy` was not examined. All solution approaches were tested with a maximum solution time of 60 seconds and $c_{k\tau} = 1$ for all $\tau \in \mathcal{T}$. The time periods were interpreted as weeks or days, with the interval length Δ set to 4 weeks and 30 days, respectively. The maximum project duration is set to $\lceil 1.25 \cdot ES_{n+1} \rceil$. The experiments were made under Windows 11 with an Intel Core i9-11900K processor at 3.5 GHz and 64 GB RAM. The results show that in most cases a simple multi-start construction heuristic is capable to find a better solution than Gurobi does. In Table 1, the gap for Gurobi and the gap between the Gurobi result and the heuristic approach are shown. A negative value for the heuristics means that the heuristic solution is better. It can be seen that a bigger interval length leads to bigger gaps for Gurobi. The heuristic with a `random` activity choice and a `greedy` start time selection (RG) performs best for the instances with an interval length of 30.

Table 1. Average gaps [%] for different selection strategies (RR: `random-random`, RG: `random-greedy`, RRW: `random-roulette wheel`, GR: `greedy-random`, GRW: `greedy-roulette wheel`) and instances (size-interval length).

Instance	**Gurobi**	**RR**	**RG**	**RRW**	**GR**	**GRW**
50-4	13.58	-2.64	-1.58	-2.67	-2.58	-2.48
100-4	23.79	-11.47	-10.56	-11.33	-11.49	-11.39
200-4	21.95	-9.55	-9.36	-9.44	-9.67	-9.50
50-30	56.91	-14.49	-15.51	-14.43	-13.43	-13.19
100-30	71.14	-31.44	-33.07	-31.30	-31.42	-31.25
200-30	66.64	-31.38	-34.03	-31.27	-31.60	-31.32

6 Outlook

This work presents a multi-start construction heuristic with different selection strategies for the novel multi-interval RIP. Further research could explore more sophisticated methods like genetic algorithms or local search. Moreover, the problem formulation could be extended by including interval-dependent interval length Δ_τ and the impact of the interval-dependent cost $c_{k\tau}$ could be examined.

References

Kreter, Stefan, Schutt, Andreas, Stuckey, Peter J, and Zimmermann, Jürgen, 2018, "Mixed-integer linear programming and constraint programming formulations for solving resource availability cost problems", *European Journal of Operational Research*, Vol. 266, pp. 472-486.

Pritsker, A Alan B, Waiters, Lawrence J, and Wolfe, Philip M, 1969, "Multiproject scheduling with limited resources: A zero-one programming approach", *Management science*, Vol. 16, pp. 93-108.

Schwindt, Christoph, 1998, "Generation of resource-constrained project scheduling problems subject to temporal constraints", *Report WIOR-543*, Institute for Economic Theory and Operations Research, University of Karlsruhe.

Best Student Paper Award #3

Maximizing the project's net present value with per-unit earliness and tardiness penalties

Lena Sophie Wohlert[1], Jürgen Zimmermann[1]

Clausthal University of Technology, Germany
lena.sophie.wohlert@tu-clausthal.de, juergen.zimmermann@tu-clausthal.de

Keywords: Project Scheduling, Net present value, Earliness-tardiness.

1 Introduction

For long-term, capital-intensive projects, the net present value is an important performance measure, and is commonly computed based on cash flows associated with the project activities. However, additional per-unit earliness penalties can arise from on-site storage or holding cost of materials or equipment. Furthermore, per-unit tardiness penalties may be contractually agreed on for a late delivery. In the literature, Afshar-Nadjafi and Shadrokh (2008) and Khoshjahan *et al.* (2013) study project scheduling problems with the objective to minimize the sum of discounted per-unit earliness and tardiness penalties. In this paper, we extend this objective by incorporating discounted cash flows. For projects with general temporal constraints, we derive structural properties and propose efficient solution approaches.

2 Problem description

A project consists of a set of n real activities, augmented by a fictitious project start 0 and a project completion $n+1$. Each activity i requires a processing time $p_i \in \mathbb{N}$, with $p_0 = p_{n+1} = 0$. The start times $S_i \in \mathbb{N}$ of all activities together form a schedule $S = (S_0, S_1, \ldots, S_{n+1})$., where the project start is fixed at $S_0 = 0$. General temporal constraints $S_j - S_i \geq \delta_{ij}$ (with $\delta_{ij} \in \mathbb{Z}$) restrict the start times, with the maximum duration $\overline{d}$ being represented as a maximum time lag between S_{n+1} and S_0. The project is modelled as an activity-on-node network $N = (V, E, \delta)$, where arcs E represent the given temporal constraints. The longest distance d_{ij} from activity i to activity j in in the project network N can be determined using a longest path algorithm. Using these distances, the earliest start time for each activity $i \in V$ is given by $ES_i = d_{0i}$, and the latest start time by $LS_i = -d_{i0}$. The earliest and latest completion times are $EC_i = ES_i + p_i$ and $LC_i = LS_i + p_i$, respectively, defining the set of feasible completion times $W_i^C = \{EC_i, \ldots, LC_i\}$.

We assume that a cash flow c_i^F, which can be either an inflow or an outflow, occurs when an activity i is completed. Each activity i may also have a due date $d_i \in W_i^C$. Then, a per-unit earliness penalty $c_i^E \geq 0$ applies for each time period activity i is finished before its due date d_i, and a per-unit tardiness penalty $c_i^T \geq 0$ applies for each period of delay. Payments are discounted to the project start at $t = 0$ using the one period discount factor $e^{-\alpha}$. The problem is formulated as follows:

$$\begin{aligned}
\max \quad & f(S) = \sum_{i \in V} c_i^F e^{-\alpha(S_i + p_i)} - \sum_{i \in V} \sum_{t=S_i+p_i}^{d_i-1} c_i^E e^{-\alpha t} - \sum_{i \in V} \sum_{t=d_i+1}^{S_i+p_i} c_i^T e^{-\alpha t} \\
\text{s.t.} \quad & S_j - S_i \geq \delta_{ij} \quad \forall \langle i, j \rangle \in E \\
& S_0 = 0 \\
& S_i \in \mathbb{N} \qquad \forall i \in V
\end{aligned}$$

3 Mathematical model formulations

A pulse-based model in which x_{it} equals 1 if activity i ends at time t, and 0 otherwise, is formulated as follows (Khoshjahan *et al.* 2013):

$$\max \quad \sum_{i\in V} c_i^F \sum_{t\in W_i^C} e^{-\alpha t} x_{it} - \sum_{i\in V} c_i^E \sum_{t=EC_i}^{d_i-1} x_{it} \sum_{\tau=t}^{d_i-1} e^{-\alpha\tau} - \sum_{i\in V} c_i^T \sum_{t=d_i+1}^{LC_i} x_{it} \sum_{\tau=d_i+1}^{t} e^{-\alpha\tau} \quad (1)$$

$$\text{s.t.} \quad \sum_{t\in W_i^C} x_{it} = 1 \qquad \forall i \in V \quad (2)$$

$$\sum_{t\in W_j^C} t x_{jt} - \sum_{t\in W_i^C} t x_{it} \geq \delta_{ij} + p_j - p_i \qquad \forall \langle i,j\rangle \in E \quad (3)$$

$$x_{it} \in \{0,1\} \qquad \forall i \in V, \forall t \in W_i^C \quad (4)$$

We also introduce a step-based model for this problem, where z_{it} is equal to 1 if activity i is completed before or at time t, and 0 otherwise (Artigues *et al.* 2014).

$$\max \quad \sum_{i\in V} c_i^F z_{i,EC_i} e^{-\alpha EC_i} + \sum_{i\in V} c_i^F \sum_{t=EC_i+1}^{LC_i} (z_{it} - z_{i,t-1}) e^{-\alpha t} - \sum_{i\in V} c_i^E \sum_{t=EC_i}^{d_i-1} e^{-\alpha t} z_{it}$$
$$- \sum_{i\in V} c_i^T \sum_{t=d_i+1}^{LC_i} e^{-\alpha t}(1 - z_{i,t-1}) \quad (5)$$

$$\text{s.t.} \quad z_{it} - z_{j,t+\delta_{ij}+p_j-p_i} \geq 0 \qquad \forall \langle i,j\rangle \in E, \forall t \in \{ES_j - \delta_{ij} + p_i, \ldots, LC_i\} \quad (6)$$

$$z_{i,t+1} \geq z_{it} \qquad \forall i \in V, \forall t \in W_i^C \setminus \{LC_i\} \quad (7)$$

$$z_{i,LC_i} = 1 \qquad \forall i \in V \quad (8)$$

$$z_{it} \in \{0,1\} \qquad \forall i \in V, \forall t \in W_i^C \quad (9)$$

The objective (5) uses that activity i ends at time t when $z_{it} - z_{i,t-1} = 1$. A per-unit earliness penalty applies at time $t < d_i$ if $z_{it} = 1$ and a per-unit tardiness penalty applies at time $t > d_i$ if activity i was not completed at the previous point in time $t-1$ ($z_{i,t-1} = 0$). Inequalities (6) enforce the general temporal constraints, while (7) ensure that if $z_{it} = 1$, the same applies for all $\tau > t$. Constraints (8) guarantee that activity i is completed by its latest completion time.

4 Structural properties and exact solution approach

The net present value objective $f^{NPV}(S) = \sum_{i\in V} c_i^F e^{-\alpha(S_i+p_i)}$ is sum-separable and for a binary direction $z \in \{0,1\}^{n+2}$ and a step-size $\lambda \geq 0$, the directional derivative is given by $\nabla f_z(S + \lambda z) = e^{-\alpha\lambda} \nabla_z f(S)$. Thus, f^{NPV} is either strictly increasing or decreasing, or constant for any binary direction z. This property is called binary monotonicity.

We now examine whether this property also holds for f, which includes additional per-unit earliness and tardiness penalties. Due to the integrality constraints on the start times, we only consider integer step sizes $\lambda \in \mathbb{N}$. Moreover, the binary direction vector z_C is defined based on a subset $C \subseteq V$ as follows:

$$z_C = \begin{cases} z_j = 1, & \text{if } j \in C \\ z_j = 0, & \text{else.} \end{cases}$$

For a schedule S, we define $\varepsilon = \{i \in C \mid S_i < d_i - p_i\}$ and $\tau = \{i \in C \mid S_i > d_i - p_i\}$. Assuming that for each $i \in \varepsilon$, the updated start time satisfies $S_i + \lambda z_{C,i} \leq d_i - p_i$, the change in the objective function from S to $S + \lambda z_C$ is given by:

$$
\begin{aligned}
&\Delta f(\lambda) = f(S + \lambda z_C) - f(S) \\
&= \sum_{j \in C} c_j^F \left(e^{-\alpha(S_j + p_j + \lambda)} - e^{-\alpha(S_j + p_j)} \right) \\
&\quad - \sum_{i \in \varepsilon} \left(\sum_{t=S_i+p_i+\lambda}^{d_i-1} c_i^E e^{-\alpha t} - \sum_{t=S_i+p_i}^{d_i-1} c_i^E e^{-\alpha t} \right) - \sum_{i \in \tau} \left(\sum_{t=d_i+1}^{S_i+p_i+\lambda} c_i^T e^{-\alpha t} - \sum_{t=d_i+1}^{S_i+p_i} c_i^T e^{-\alpha t} \right) \\
&= \sum_{j \in C} c_j^F e^{-\alpha(S_j+p_j)} \left(e^{-\alpha\lambda} - 1 \right) + \sum_{i \in \varepsilon} c_i^E \sum_{t=S_i+p_i}^{S_i+p_i+\lambda-1} e^{-\alpha t} - \sum_{i \in \tau} c_i^T \sum_{t=S_i+p_i+1}^{S_i+p_i+\lambda} e^{-\alpha t} \\
&= \sum_{j \in C} c_j^F e^{-\alpha(S_j+p_j)} \left(e^{-\alpha\lambda} - 1 \right) + \sum_{i \in \varepsilon} c_i^E e^{-\alpha(S_i+p_i)} \sum_{t=0}^{\lambda-1} e^{-\alpha t} - \sum_{i \in \tau} c_i^T e^{-\alpha(S_i+p_i)} \sum_{t=1}^{\lambda} e^{-\alpha t}
\end{aligned}
$$

Let us distinguish two cases for $\lambda = 1$: either $\Delta f(1) > 0$ or $\Delta f(1) \leq 0$. We begin with the case $\Delta f(1) > 0$. Assuming $\Delta f(\lambda) > 0$ for some $\lambda \geq 1$, the question arises whether the same also holds for $\Delta f(\lambda + 1)$ (under the condition that for each activity $i \in \varepsilon$, the updated start time still satisfies $S_i + (\lambda + 1) z_{C,i} \leq d_i - p_i$).

$$
\begin{aligned}
&\Delta f(\lambda + 1) \\
&= \sum_{j \in C} c_j^F e^{-\alpha(S_j+p_j)} \left(e^{-\alpha(\lambda+1)} - 1 \right) \\
&\quad + \sum_{i \in \varepsilon} c_i^E e^{-\alpha(S_i+p_i)} \left(\sum_{t=0}^{\lambda-1} e^{-\alpha t} + e^{-\alpha\lambda} \right) - \sum_{i \in \tau} c_i^T e^{-\alpha(S_i+p_i)} \left(\sum_{t=1}^{\lambda} e^{-\alpha t} + e^{-\alpha(\lambda+1)} \right)
\end{aligned}
$$

With

$$
c_j^F e^{-\alpha(S_j+p_j)} \left(e^{-\alpha\lambda} e^{-\alpha} - 1 \right) = c_j^F e^{-\alpha(S_j+p_j)} \left(e^{-\alpha\lambda} - 1 \right) + c_j^F e^{-\alpha(S_j+p_j)} e^{-\alpha\lambda} \left(e^{-\alpha} - 1 \right)
$$

the following holds:

$$
\begin{aligned}
&\Delta f(\lambda + 1) \\
&= \sum_{j \in C} c_j^F e^{-\alpha(S_j+p_j)} (e^{-\alpha\lambda} - 1) + \sum_{i \in \varepsilon} c_i^E e^{-\alpha(S_i+p_i)} \sum_{t=0}^{\lambda-1} e^{-\alpha t} - \sum_{i \in \tau} c_i^T e^{-\alpha(S_i+p_i)} \sum_{t=1}^{\lambda} e^{-\alpha t} \\
&\quad + e^{-\alpha\lambda} \left(\sum_{j C} c_j^F e^{-\alpha(S_j+p_j)} (e^{-\alpha} - 1) + \sum_{i \in \varepsilon} c_i^E e^{-\alpha(S_i+p_i)} - \sum_{i \in \tau} c_i^T e^{-\alpha(S_i+p_i+1)} \right) \\
&= \Delta f(\lambda) + e^{-\alpha\lambda} \left(\sum_{j \in C} c_j^F e^{-\alpha(S_j+p_j)} (e^{-\alpha} - 1) + \sum_{i \in \varepsilon} c_i^E e^{-\alpha(S_i+p_i)} - \sum_{i \in \tau} c_i^T e^{-\alpha(S_i+p_i+1)} \right) \\
&= \Delta f(\lambda) + e^{-\alpha\lambda} \Delta f(1) > 0
\end{aligned}
$$

So if $\Delta f(1) > 0$, the function is monotonically increasing. Likewise, if $\Delta f(1) \leq 0$, the function is monotonically decreasing. Therefore, the extended objective function remains binary monotone, though only on the same set ε. Once an activity i no longer incurs earliness penalties, i.e. $S_i = d_i - p_i$, the monotonicity may change. From this point on, the

positive earliness penalty term for activity i is absent from Δf, so the monotonicity cannot switch from decreasing to increasing. Moreover, f remains sum-separable. Therefore, the steepest ascent approach to maximize f^{NPV}, introduced by Schwindt and Zimmermann (2001), can be adapted here as an exact solution method. For the f^{NPV}, the algorithm starts with the earliest start schedule and then a steepest ascent direction based on the derivatives is determined. Due to the binary monotonicity of f^{NPV}, this direction remains unchanged for all non-negative step sizes. Thus, step sizes are only limited by the temporal constraints. After updating the respective start times with the chosen step size, a new steepest ascent direction is identified. Starting from the earliest schedule ensures that step sizes are always non-negative i.e. the start times of all activities are non-decreasing over the iterations. In order to adapt the steepest ascent algorithm to the problem at hand, $\Delta f(1)$ must be evaluated instead of the derivatives. Furthermore, step sizes are restricted by both temporal constraints and activity due dates.

5 Experiments

The problem instances are based on the benachmark test sets UBO[1] for the RCPSP/max. The maximum project duration is set to $\lceil 1.5 \cdot ES_{n+1} \rceil$. Each activity is assigned a cash flow $c_i^F \in \{-1000, \ldots, 5000\}$, with 25% of activities having a cash inflow. 10 % of activities are assigned a due date d_i, as well as per unit earliness and tardiness penalties $c_i^E, c_i^T \in \{1, \ldots, 5\}$. Payments are discounted with $\alpha = 0.01$. The solution approaches are implemented in C++ with a 300-second runtime limit. The mathematical models are solved by Gurobi.

Table 1. Preliminary results

Instances	Mathematical formulations				Adapted steepest ascent approach
	Pulse-based		Step-based		
n	∅gap[%]	$\varnothing t_{opt}[s]$	∅gap[%]	$\varnothing t_{opt}[s]$	$\varnothing t_{opt}[s]$
50	0	2.69	0	0.22	0.01
100	0.23	81.17	0	1.44	0.02
200	29.10	284.47	0	21.09	0.05

Of the mathematical models solved by Gurobi, the step-based formulation achieves a shorter average runtime and a smaller average gap for larger instances than the pulse-based (x_{it}) formulation. The adapted steepest ascent approach utilising the identified structural properties achieves the best overall runtime and terminates within the time limit for all instances. Consequently, it always obtains optimal solutions.

References

Afshar-Nadjafi B., S. Shadrokh, 2008, "An algorithm for the weighted earliness-tardiness unconstrained project scheduling problem", *Journal of Applied Sciences*, Vol. 8, pp. 1651–1659.

Artigues C., O. Koné and P. Lopez, M. Mongeau, 2014, "Mixed-integer linear programming formulations", In *Handbook on project management and scheduling*, Vol. 1, pp. 17–41.

Khoshjahan Y., A. A. Najafi and B. Afshar-Nadjafi, 2013, "Resource constrained project scheduling problem with discounted earliness–tardiness penalties: Mathematical modeling and solving procedure", *Computers & Industrial Engineering*, Vol. 66, pp. 293–300.

Schwindt C., J. Zimmermann, 2001, "A steepest ascent approach to maximizing the net present value of projects", *Mathematical Methods of Operations Research*, Vol. 53, pp. 435–450.

[1] https://www.wiwi.tu-clausthal.de/en/ueber-uns/abteilungen/betriebswirtschaftslehre-insbesondere-produktion-und-logistik/research/research-areas/project-generator-progen/max-and-psp/max-library/single-mode-project-duration-problem-rcpsp/max

Mathematical Programming for Workload-Balanced Scheduling of Resource-Constrained Projects

Loris Trotter, Nina Ackermann, Tamara Bigler and Norbert Trautmann

University of Bern, Switzerland
loris.trotter@unibe.ch, nina.ackermann@unibe.ch,
tamara.bigler@unibe.ch, norbert.trautmann@unibe.ch

Keywords: Workload-Balanced Project Scheduling, Mixed Integer Linear Programming.

1 Introduction

Projects are an essential part of business operations, representing a significant share of overall economic activities (Schoper *et al.* 2018). According to Klein (2000), a project is "a one-time activity with specific objectives which has to be realized in a certain period of time using a limited number of resources." In practice, project resources frequently represent teams of people. Consequently, project success often depends on high team performance, which in turn requires strong collaboration and cohesion (Yang *et al.* 2011). A significant factor in achieving this is a well-balanced contribution from all team members (Hoegl and Gemuenden 2001). The allocation of activities among the team members is conducted in the planning phase of a project, in which determining a suitable project schedule is a pivotal task (Klein 2000).

We consider a project that consists of multiple activities and requires several types of resources. The objective is to determine a schedule with optimal workload balance across the resource types. To this end, we propose to minimize the schedule's total workload imbalance, measured as the sum of the differences between the maximum and minimum unit workloads for each resource type. Furthermore, all activities must be completed within a prescribed time horizon, which results from the client's project deadline. The schedule must also be feasible with respect to the prescribed resource capacities and precedence relations among the activities.

In this paper, we outline a novel MIP-based approach for workload-balanced scheduling of resource-constrained projects with a fixed maximum duration. We implement this approach as an extension of the mixed-integer linear programming (MIP) formulation proposed by Rihm and Trautmann (2017). Our computational results indicate that this approach consistently yields project schedules with a markedly better workload balance compared to schedules devised without workload balance consideration.

Most MIP formulations for project scheduling (for an overview, see Artigues *et al.* 2015) focus on minimizing project duration while accounting for resource capacities and precedence relations among activities. Our work contributes to this research by demonstrating that the workload balance in project schedules can often be improved considerably, which fosters stronger team performance. The objective of workload balancing has been studied in other scheduling contexts such as parallel machine scheduling (cf., e.g., Cossari *et al.* 2013, Ouazene *et al.* 2014), but has not been considered in the project scheduling literature.

The remainder of this paper is structured as follows. In Section 2, we describe the planning problem in detail. In Section 3, we outline our MIP-based approach and report on the results of our computational experiment. In Section 4, we conclude the paper and provide an outlook on future research.

2 Planning problem

We consider a project that consists of n activities. We represent the start and finish of the project by introducing two fictitious activities, 0 and $n+1$, and denote the set of all activities by $V := \{0, 1, \ldots, n, n+1\}$. Each activity $i \in V$ has a duration p_i and requires renewable resources during execution. We denote the set of resource types by R. For each resource type $k \in R$, R_k denotes the capacity, i.e. the maximum number of units that can be used simultaneously, and r_{ik} denotes the number of units of resource type k required by activity i. The two fictitious activities 0 and $n+1$ have a duration of zero and do not require any resources. We denote the set of prescribed precedence relations by E: for each pair of activities $(i, j) \in E$, activity i must be completed before activity j can start. The project must be completed within the prescribed time horizon T, specified by the client's project deadline. The objective is to optimize the workload balance across the resource types. A common measure of workload balance in machine scheduling is the difference between the maximum and minimum workload across machines (cf., e.g., Cossari *et al.* 2013, Ouazene *et al.* 2014). Following this concept, we define the workload imbalance of resource type $k \in R$ as $\delta_k = \overline{r}_k - \underline{r}_k$, where $\overline{r}_k$ and $\underline{r}_k$ denote the maximum and minimum workloads across the units of resource type k. An optimal solution is a project schedule, i.e. a start time for each activity, such that workload imbalance is minimized, the project is completed within the time horizon T, and all precedence relations and resource capacities are satisfied.

We illustrate the planning problem through the following example. Consider a project with $n = 7$ activities, depicted as an activity-on-node network in Figure 1 (left). The project involves a single resource type $k = 1$, representing a team of $R_1 = 5$ team members. For example, activity $i = 2$ has a duration of $p_2 = 3$ and requires $r_{2,1} = 1$ team member. The arcs in the activity-on-node network represent the set of precedence relations E. For instance, activity $i = 1$ must be completed before activity $j = 2$ can start. The project must be completed within the prescribed time horizon $T = 16$. Figure 1 (right) illustrates a feasible schedule for this example project. The maximum workload of resource type $k = 1$ is $\overline{r}_1 = 12$ (units l = 1 and l = 2), and the minimum workload is $\underline{r}_1 = 4$ (unit l = 5). Therefore, the workload imbalance is $\delta_1 = 12 - 4 = 8$.

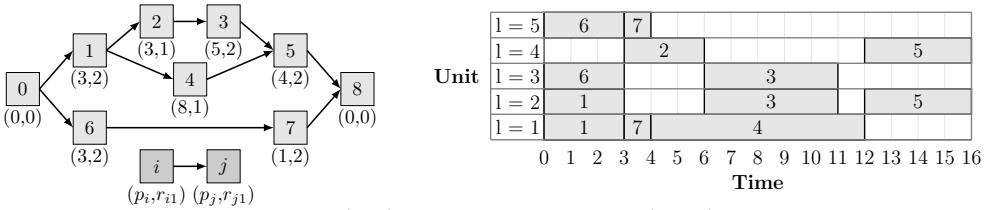


Fig. 1: Activity-on-node network (left) and feasible schedule (right) for the example project

3 MIP-based solution approach

In this section, we outline our novel MIP-based solution approach for workload-balanced scheduling and present the results of our computational analysis.

3.1 Overview

To optimize workload balance, the objective function to be minimized is the sum $\sum_{k \in R} \delta_k$ of the workload differences across all resource types. Analogously to Ackermann *et al.* (2024), the constraints for computing δ_k are based on the CTAB model proposed by Rihm and Trautmann (2017). This unit-based continuous-time formulation has demonstrated a solid performance and inherently incorporates information on the workload of individual resource units, which is essential for determining the workload balance.

Our formulation focuses on creating a feasible schedule that meets the specified deadline. To this end, we introduce an additional constraint to ensure that the project duration does not exceed the time horizon T. Depending on the value of T, it might be challenging to devise a feasible schedule; therefore, we propose to proceed as follows. First, the CTAB model with the objective of minimizing the project duration, but without consideration of workload balancing, is applied, and the solver is stopped as soon as a feasible schedule has been found according to which the project is completed within the given time horizon. Then, the extended CTAB model for workload balancing is applied, using this schedule as an (incomplete) warm start solution.

We illustrate the effect of workload balancing in Figure 2 using the example project from Section 2. The schedule generated with workload balancing is shown in dark gray, while the light gray schedule corresponds to the solution presented in Figure 1, obtained without workload balancing. Both schedules represent feasible solutions to the planning problem: the project is completed within the time horizon ($t = 16$), all precedence relations are satisfied, and the capacities of all renewable resources are respected. In the schedule with workload balancing, the maximum workload is $\overline{r}_1 = 9$ and the minimum workload is $\underline{r}_1 = 8$, resulting in a difference of $\delta_1 = 1$. Without workload balancing, the maximum workload is $\overline{r}_1 = 12$, the minimum workload is $\underline{r}_1 = 4$, and the difference is $\delta_1 = 8$. Although both schedules are feasible solutions, the workload is distributed considerably more evenly across units when workload balancing is applied. Furthermore, Figure 2 shows that the improvement in workload balance cannot simply be achieved by moving activities between units, but also requires adjusting their start times (see, for example, activity 6).

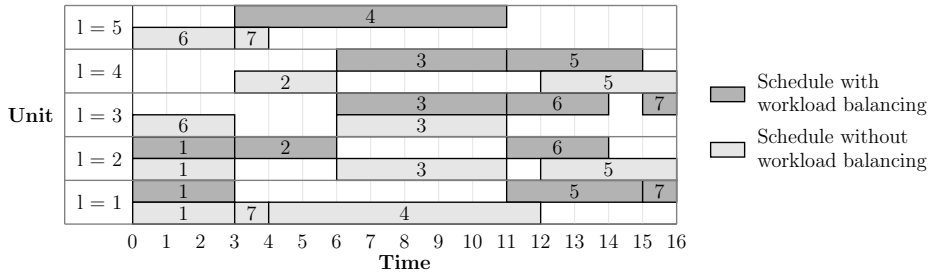


Fig. 2: Comparison of project schedules for the example project

3.2 Computational results

In this subsection, we evaluate the performance of the novel MIP-based workload balancing approach outlined in Subsection 3.1 by benchmarking it against the CTAB model without consideration of workload balancing. For this comparative analysis, we used the J30 test set from PSPLIB (Kolisch and Sprecher 1996), comprising 480 project instances, each with $n = 30$ activities and up to $k = 4$ resource types. For each instance, we set the project time horizon to 110% of the optimal makespan for the project-duration minimization problem, assuming that the prescribed deadline includes a reasonable amount of buffer time. Both MIP models were implemented in Python 3.13 and solved with Gurobi 13.0.0 on an Apple M4 Max (128 GB RAM). For each phase of the solution approach, we set the solver time limit to 60 seconds per instance and limited the number of threads to two.

Table 1 summarizes the comparison results. Following the procedure outlined in Subsection 3.1, the first row presents the aggregated results after the first phase, in which the CTAB model was applied without workload balancing. For all instances, a feasible schedule with a project completion within the prescribed time horizon was found within the 60-second time limit. The second row represents the schedules after the second phase, in

which the extended CTAB model for workload balancing was applied. Applying this model reduced the workload imbalance in all instances, decreasing the average total workload imbalance by 93.5% compared to the schedules before workload balancing. Optimality was achieved in 445 instances, often within a few seconds. For the remaining 35 instances, no proven optimal solution was found within the solver time limit of 60 seconds; however, also in these instances, the workload imbalance was reduced by 92.5% on average compared to before workload balancing.

Table 1: Comparison of computational results

Schedules	**Avg. $\sum_k \delta_k$**	**Avg. runtime**
Before workload balancing	103.94	1.36
After workload balancing	6.79	11.22

4 Conclusion

In this paper, we analyzed a novel MIP-based approach to workload-balanced scheduling of resource-constrained projects with a given time horizon. We illustrated that the proposed approach generates schedules with considerably more balanced workloads than schedules generated without consideration of workload balancing.

For future research, we propose to develop a weighted goal programming approach that simultaneously considers both the conventional project-scheduling criterion of minimizing project duration and the newly introduced criterion of optimizing workload balance.

References

Ackermann, N., T. Bigler and N. Trautmann, 2024, "Workload-balancing constraints in a continuous-time integer programming formulation for the resource-constrained project scheduling problem", in *Proceedings of the 2024 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Bangkok, Thailand, pp. 805–808.

Artigues, C., O. Koné, P. Lopez and M. Mongeau, 2015, "Mixed-Integer Linear Programming Formulations", in C. Schwindt and J. Zimmermann (eds.), *Handbook on Project Management and Scheduling, Vol. 1*, *International Handbooks on Information Systems*, Springer, Cham.

Cossari, A., J. Ho, G. Paletta and A. Ruiz-Torres, 2013, "Minimizing workload balancing criteria on identical parallel machines", *Journal of Industrial and Production Engineering*, Vol. 30(3), pp. 160–172.

Hoegl, M. and H. Gemuenden, 2001, "Teamwork quality and the success of innovative projects: a theoretical concept and empirical evidence", *Organization Science*, Vol. 12(4), pp. 435–449.

Klein, R., 2000, *Scheduling of Resource-Constrained Projects*, Springer, New York.

Kolisch, R. and A. Sprecher, 1996, "PSPLIB – A project scheduling problem library: OR Software – ORSEP Operations Research Software Exchange Program", *European Journal of Operational Research*, Vol. 96(1), pp. 205–216.

Schoper, Y., A. Wald, H. Ingason and T. Fridgeirsson, 2018, "Projectification in western economies – a comparative study of Germany, Norway and Iceland", *International Journal of Project Management*, Vol. 36(1), pp. 71–82.

Rihm, T. and N. Trautmann, 2017, "An assignment-based continuous-time MILP model for the resource-constrained project scheduling problem", in *Proceedings of the 2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore, pp. 35–39.

Ouazene, Y., F. Yalaoui, H. Chehade and A. Yalaoui, 2014, "Workload balancing in identical parallel machine scheduling using a mathematical programming method", *International Journal of Computational Intelligence Systems*, Vol. 7(sup1), pp. 58–67.

Yang, L., C. Huang and K. Wu, 2011, "The association among project manager's leadership style, teamwork and project success", *International Journal of Project Management*, Vol. 29(3), pp. 258–267.

16-16 Apr 2026

L - Machine learning and Hybrid methods

Duration forecasting using project complexity and sensitivity in a Bayesian Network model

Izel Unsal Altuncan[1], Mario Vanhoucke[1,2,3]

[1] Ghent University, Belgium
izel.unsalaltuncan@ugent.be
[2] Vlerick Business School, Belgium
[3] University College London,UK
mario.vanhoucke@ugent.be

Keywords: Duration Forecasting, Bayesian Network, Project Complexity

1 Introduction

Project managers and researchers need accurate duration forecasts to take actions for keeping the projects on track. Existing forecasting approaches are split into *static methods* that rely on initially estimated activity risk profiles and *dynamic methods* that rely on the progress data and update the forecasts during project execution. The static methods has no adaptability based on the progress, while the dynamic methods often assume the complexity of the finished part of the project is the same with the remaining. These shortcomings leave a gap for a forecasting method that models risk profiles and project complexity, and also updates as progress data becomes available. We address this gap with an evolving Bayesian Network model (BN), performed in three approaches. We integrate network and resource complexity indicators from *Project Scheduling* with time, cost and resource sensitivity metrics from *Schedule Risk Analysis* into a probabilistic forecasting framework that works both before the project start and during execution. Figure 1 summarizes the three-approach evolution as follows. Approach 1 builds a Bayesian Network model with network complexity variables and time and cost sensitivity metrics. This approach ignores resource limitations and applies the model statically before the start. Approach 2 adds resource-related variables while keeping the static framework and Approach 3 applies the resource-extended model dynamically during execution.

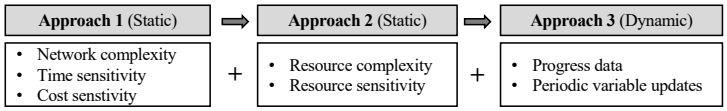


Fig. 1. Summary of the model variables used in each approach

The Bayesian Network model represents causal dependencies among the complexity and sensitivity variables. Making use of these variables, the model

generates predictions for the probabilistic distributions of the project duration. All three approaches follow the same underlying logic following five phases. The specifics of each approach is given in the following sections. Phase 1 specifies the theoretical model by defining variables and their causal dependencies. Not all three approaches rely on the same set of variable. Approach 1 is the most restrictive one, only incorporating the grey variables in Figure 2, while the two other approaches are more extensive and make use of all variables in the figure. Variables (ellipses) are defined as weighted sums of associated indicators (rectangles). The indicators follow [Van Eynde and Vanhoucke, 2022] for network and resource related complexity, [Vanhoucke, 2010] for *Time*, *Cost* and *Resource Sensitivity*, and [Song et al., 2021] for *Resource Time Sensitivity*. Phase 2 simulates artificial project instances to imitate progress under multiple duration uncertainty settings. Phase 3 validates the theoretical model on the training split of the project data from Phase 2 using Structural Equation Modeling (SEM). This phase leads to the exclusion of some variables and retention of only the significant metrics for the retained variables. Phase 4 trains the validated model also on the training split, using Bayesian learning to estimate parameters of the probabilistic causality. Phase 5 feeds the trained model with the variable values of instances in the test split and predict the expected project duration.

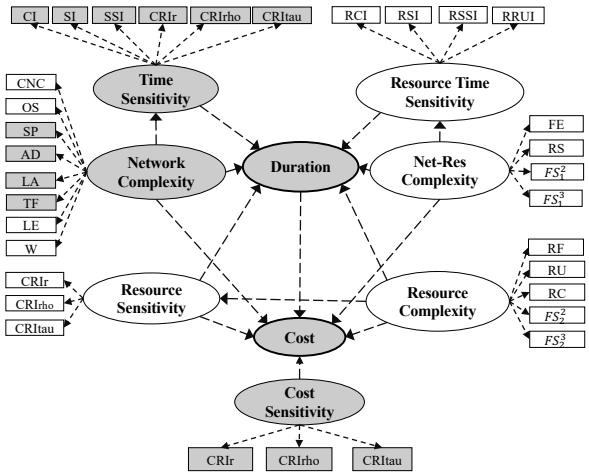


Fig. 2. Evolution of the theoretical model

2 Approach 1. Static forecasting without resources

The first approach tests whether the Bayesian Network model built on network complexity and time and cost sensitivity variables improve the static duration

forecasts of the benchmarks. Forecasting accuracy is compared against Monte Carlo simulations (MC) and machine-learning algorithms (ML). The results show that the Bayesian Network model delivers the most accurate forecasts when duration uncertainty is low and moderate, yet forecasts are less accurate when actual deviations from the baseline schedule are significantly large (leftmost pane of Table 1). This approach validates the static use of the Bayesian Network model before the project execution begins and highlights the need to improve accuracy under higher uncertainty settings.

3 Approach 2. Static forecasting with resources

This approach extends the Bayesian Network model by adding variables to represent resource complexity and resource sensitivity to test the accuracy when forecasting resource-constrained projects. We simulate such project instances under varying levels of duration uncertainty and execution flexibility, which is defined by the percentage of activities that can start earlier than plan. The extended model is tested on projects with higher uncertainty to investigate whether the shortcoming of the first study is well addressed. We evaluate the performance of the extended model by comparing with the initial model from Approach 1 and with the same benchmarks. This approach yields three main results. First, the resource-extended model improves accuracy over the initial model, with the largest improvement for projects having strict resource constraints. Second, it outperforms the benchmarks under higher levels of uncertainty, especially when execution flexibility is estimated accurately during model training. Third, the impact of execution flexibility depends on the network topology. In parallel projects, our model is more accurate as flexibility increases and it performs best when the training flexibility matches the test flexibility (middle pane of Table 1). In serial projects, flexibility has almost no impact because possible early activity starts are limited since the precedence constraints are strict.

4 Approach 3. Dynamic forecasting with resources

The third approach applies the resource-extended model dynamically during project execution. The model is performed at nine tracking periods from 10% to 90% completion. At each period the project progress is observed, finished activities are removed from the activity network and the variables are re-calculated on the remaining network. We evaluate the accuracy across different levels of network topology and resource scarcity, duration uncertainty and different policies for updating the input variables. The accuracy is benchmarked against dynamic methods based on the earned value (EV). The results show that the dynamic Bayesian Network model is most accurate for more complex projects having parallel networks under high resource scarcity on average over all tracking periods. However, in low complexity projects with serial networks and low scarcity, benchmarks outperform our model over all periods (rightmost pane of Table 1). Finally, the initial results indicate that updating all variables at each period is not optimal, as selective updating yields higher forecasting accuracy.

5 Conclusions

This study presents a continuing research for forecasting project duration and the results obtained so far. Although the research is not yet complete, the results highlight three findings. First, quantitative metrics from scheduling and risk analysis are valid indicators of project duration performance and can be used for forecasting. Second, forecasts for resource-constrained projects should account for the resource-related characteristics. Third, these quantitative metrics change during execution and updating them periodically yields more accurate forecasts.

	Approach 1 (uncertainty)			Approach 2 (flexibility)			Approach 3 (complexity)		
	Low	Medium	High	Low	Medium	High	Low	Medium	High
BN	2.4	5.4	10.1	5.8	5.8	31.0	15.5	4.9	1.1
MC	6.3	24.7	31.0	3.5	9.8	45.2	-	-	-
ML	3.7	7.3	9.2	7.4	4.9	35.7	-	-	-
EV	-	-	-	-	-	-	9.9	6.5	10.0

Table 1. Summary of the accuracy comparison in terms of % error (MAPE)

References

[Song et al., 2021] Song, J., Martens, A., and Vanhoucke, M. (2021). Using schedule risk analysis with resource constraints for project control. *European Journal of Operational Research*, 288.

[Unsal-Altuncan and Vanhoucke, 2024] Unsal-Altuncan, I. and Vanhoucke, M. (2024). A hybrid forecasting model to predict the duration and cost performance of projects with bayesian networks. *European Journal of Operational Research*, 315:511–527.

[Unsal-Altuncan and Vanhoucke, 2025a] Unsal-Altuncan, I. and Vanhoucke, M. (2025a). Duration forecasting in resource constrained projects: A hybrid risk model combining complexity indicators with sensitivity measures. *European Journal of Operational Research*. Advance online publication. Available at: https://doi.org/10.1016/j.ejor.2025.03.012.

[Unsal-Altuncan and Vanhoucke, 2025b] Unsal-Altuncan, I. and Vanhoucke, M. (2025b). A dynamic bayesian risk model for duration forecasting in resource-constrained projects. Manuscript submitted for publication.

[Van Eynde and Vanhoucke, 2022] Van Eynde, R. and Vanhoucke, M. (2022). A theoretical framework for instance complexity of the resource-constrained project scheduling problem. *Mathematics of Operations Research*, 47:3156–3183.

[Vanhoucke, 2010] Vanhoucke, M. (2010). Using activity sensitivity and network topology information to monitor project time performance. *Omega The International Journal of Management Science*, 38:359–370.

Algorithm Selection for the Resource-Constrained Project Scheduling Problem using Graph Neural Networks

Hendrik Weber and Rainer Kolisch

Technical University of Munich, Germany
hendrik.weber@tum.de, rainer.kolisch@tum.de

Keywords: resource-constrained project scheduling problem, algorithm selection, machine learning, heuristics

1 Introduction and problem description

The Resource-Constrained Project Scheduling Problem (RCPSP) is known to be strongly NP-hard and solving even small instances using exact methods can be computationally prohibitive. Consequently, a substantial portion of research efforts has focused on developing approximate solution methods that produce high-quality solutions within reasonable computation times.

Newly proposed solution methods are typically evaluated against standardized instance sets to ensure robustness across a wide range of problem characteristics. While such benchmarking provides valuable aggregate performance metrics, it is widely recognized that no single algorithm performs best across all instances. Some algorithms may perform exceptionally well on specific subsets of instances, while performing poorly on others.

This phenomenon, known as algorithm complementarity, provides a strong motivation for algorithm selection strategies. The algorithm selection problem (ASP) has been originally proposed by Rice (1976). Several studies in the context of the RCPSP have shown that supervised learning techniques can effectively address the algorithm selection problem by learning mappings from instance features to algorithm performance. However, a potential shortcoming of these aggregating features is that they compress complex information of a problem instance into low-dimensional representations. This mapping may not be injective and different problem instances may have identical feature representations, thus obscuring meaningful structural differences that distinguish instances.

To address this limitation, graph neural networks (GNNs) have emerged as a promising tool for learning directly over combinatorial structures. Unlike traditional machine learning models that require fixed-size tabular inputs, GNNs can process variable-sized inputs, making them suitable for learning over scheduling problem instances of differing sizes and complexities. Here, we utilize the canonical Activity-on-Node representation of scheduling problems, where the precedence graph serves as the computation graph of the machine learning model.

We formulate the algorithm selection problem for the RCPSP as a supervised learning task, focusing on GNN models to predict algorithm performance for given problem instances. The initial algorithm portfolio consists of 22 static priority rules, used in combination with the serial schedule generation scheme. In our computational study, we investigate the impact of different GNN architectures, loss functions, and the inclusion (or omission) of traditional instance characteristics in our model.

2 Related literature

Recent research includes the works of Messelis and De Causmaecker (2014), Guo *et al.* (2021), and Guo, Vanhoucke and Coelho (2023).

Messelis and De Causmaecker (2014) address the algorithm selection problem for the multi-mode resource-constrained project scheduling problem (MRCPSP), employing two different methodological perspectives: First, they treat the algorithm selection problem as a regression task, developing separate prediction models for each available solution approach and aim to directly predict the objective function value of the considered algorithm-instance pair. Second, they treat the ASP as a classification problem by labeling the training data with which algorithm performs best on any given instance.

Guo *et al.* (2021) address the ASP for the single-mode RCPSP considering an algorithm portfolio of 39 priority rules and similarly to Messelis and De Causmaecker (2014) focus their work on decision tree-based approaches. The authors frame the learning task as a multi-label classification and a multi-class classification, using the binary relevance approach and label powerset transformation, respectively.

Guo, Vanhoucke and Coelho (2023) focus on exact solution approaches for the RCPSP and introduce an algorithm configuration model that determines tree search strategy, branching order and lower bound calculations of a branch-and-bound algorithm. Especially relevant to our research is the framing of the learning task as a label ranking problem, assigning each training instance a strict ordering of the considered algorithm configurations.

Although these previous contributions in the field of algorithm selection for the RCPSP differ in what problem variant and solution approaches are considered or how the learning task is handled, they all share a similar approach for generating an informative instance feature set for their respective learning models. They mostly rely on established RCPSP instance features that have been proposed in the past. Here, we believe that a prediction model for the algorithm selection problem can benefit from GNN-based approaches, as they have shown promising results in related domains such as Reinforcement Learning for the RCPSP (Zhao *et al.* 2022) or custom GNN-based heuristics (Teichteil-Königsbuch *et al.* 2023, Verhaeghe *et al.* 2024).

3 Methodology

We formulate the ASP for the RCPSP as a supervised learning task with the objective of minimizing the average critical-path-normalized makespan, denoted as ϕ, of an unseen test set.

For the development of our algorithm selection model, we focus on GNN-based approaches and leverage the Activity-on-Node representation of RCPSP instances. Each RCPSP instance is partially defined by its precedence graph $G = (V, E)$ where the set of vertices V represents activities and the set of directed edges E describes the precedence relations between activities. Each activity $i \in V$ is associated with a duration p_i and renewable resource requirements $r_{i,k}$. This precedence graph serves as the input graph for our proposed GNN models. In order to increase the information content of the precedence graph, we extend the aforementioned attributes of activities with common activity features from the scheduling literature, e.g., earliest start time, free float or number of predecessors. Thus, each activity i is associated with a node feature vector $x_i \in \mathbb{R}^d$ that includes attributes directly from the problem instance, as well as derived features.

We denote the set of candidate algorithms, also known as algorithm portfolio, as Q. Our learning task can then be formalized as follows. We learn a mapping $F_\theta : \mathcal{G} \to \mathbb{R}^{|Q|}$, $z_G = F_\theta(G)$, i.e., a function that maps each instance's precedence graph to a logit vector over

the algorithm portfolio. Based on z_G, We then select a single priority rule that is expected to perform best on the given instance.

Prior works have mostly framed the ASP for the RCPSP as a classification problem, i.e., encoding the best-performing algorithms as a multi-hot vector and employing a cross-entropy loss function. However, this approach ignores the severity of misclassifications, which should be proportional to the makespan. An accurate model that manages to frequently identify the best-performing algorithms does not necessarily coincide with also generating the lowest average makespan for the considered instance set.

In order to investigate the impact of loss function choice, we also treat the problem as a (multi-label) classification problem, employing a cross-entropy loss function. However, we additionally frame the problem as a ranking problem to better align the loss function with the objective of achieving a low average makespan on the test instance set. Our ranking approach is based on determining regret values for each instance-algorithm pair, which then serve as the ground truth labels for the machine learning model. We denote the makespan when instance i is solved with algorithm $q \in Q$ as $C_{i,q}$ and the critical-path-based lower bound as cpm_i. The regret values $\rho_{i,q}$ are calculated as $\rho_{i,q} = \frac{C_{i,q} - \min_{t \in Q} C_{i,t}}{cpm_i}$ and directly represent the ordering of algorithms for problem instance i.

4 Computational study and preliminary results

In our computational study, we examine the performance of multiple different GNN architectures, two loss functions and different sizes of algorithm portfolios. We limit our study to the GNN models Graph Convolutional Networks (GCN) and Graph Isomorphism Networks (GIN), which mainly differ in their neighborhood aggregation schemes. Additionally, some architectural choices are embedded in a hyperparameter study. This includes the number of GNN layers, the pooling operator, dimensions of a preproccessing MLP, and whether to use residual connections or not, among others. The considered loss functions are the binary cross-entropy loss for the classification approach and a ListNet loss for the ranking approach.

A preliminary investigation of the dataset revealed that the potential improvement of using an algorithm selection approach compared to the single best-performing priority rule is rather small. We also found that the marginal benefit of using a larger algorithm portfolio quickly diminishes, as a perfect predictor that considers only a subset of eight of the 22 priority rules already manages to realize 90% of the aforementioned potential improvement. In a more realistic scenario, where the prediction model is bound to misclassify instances, a large algorithm portfolio may actually lead to a deterioration of the model's performance, as it allows for the selection of generally inferior priority rules. In contrast, a small algorithm portfolio might not include the priority rule that yields the lowest makespan on a specific instance, yet still lead to an overall better performance due to the decreased odds of misclassification. To investigate this trade-off between improvement potential and prediction accuracy, we limit the number of candidate algorithms to $|Q| \in \{2, 4, 8, 12\}$.

We assess the quality of our proposed models in terms of the the average critical-path-normalized makespan ϕ as well as the accuracy on unseen test sets. The metric ϕ is determined based on the makespans achieved by the predicted priority rules on the unseen test set P_{test} (lower is better) and calculated, for example, for the GCN model as $\phi(GCN) = \frac{1}{|P_{test}|} \sum_{i \in P_{test}} \frac{C_{i,GCN(i)}}{cpm_i}$ where $GCN(i)$ denotes the priority rule predicted with the GCN for instance i. When determining accuracy, we treat every problem instance where the prediction matches any of the best-performing priority rules as a positive sample (higher is better).

We also include the myopic approach of only using the single best priority rule (SBS), as well as an oracle predictor (VBS) in our reported results, as they provide lower and upper bounds for any viable algorithm selection model. The SBS is determined by the single priority rule that achieves the lowest average normalized makespan ϕ on the training set, while the VBS represents the hypothetical approach of always selecting the best-performing priority rule for each problem instance.

Our considered dataset consists of 13,413 RCPSP instances.

Table 1 summarizes the results for the GCN and GIN models in comparison to the oracle approach (VBS) and single best-performing priority rule approach (SBS). We find that even though our proposed models consistently outperform the single priority rule approach, they do not manage to fully leverage larger algorithm portfolios. We attribute this to the fact, that the models limit their predictions to only few algorithms, even when more candidate algorithms exist. The choice of classification or ranking approach seems to have only a small effect on the achieved makespans, however, we generally observe a slightly lower accuracy for the models trained with a ranking-based loss function.

Table 1. Results for GCN and GIN models in comparison to oracle and single priority rule

		ϕ				**Accuracy (%)**			
	$\|Q\|$	**VBS**	**GCN**	**GIN**	**SBS**	**VBS**	**GCN**	**GIN**	**SBS**
Classification	**2**	2.1063	2.1265	2.1264	2.1359	100.0	75.7	76.1	67.0
	4	2.0916	2.1259	2.1257	2.1359	100.0	60.2	60.5	52.0
	8	2.0806	2.1260	2.1263	2.1359	100.0	51.7	51.3	44.3
	12	2.0761	2.1259	2.1259	2.1359	100.0	48.8	48.9	41.4
Ranking	**2**	2.1063	2.1277	2.1273	2.1359	100.0	75.1	75.3	67.0
	4	2.0916	2.1262	2.1258	2.1359	100.0	59.6	60.1	52.0
	8	2.0806	2.1261	2.1259	2.1359	100.0	51.5	51.4	44.3
	12	2.0761	2.1258	2.1264	2.1359	100.0	48.9	48.8	41.4

References

Guo, W., Vanhoucke, M., Coelho, J. and Luo, J., 2021, "Automatic detection of the best performing priority rule for the resource-constrained project scheduling problem", *Expert Systems with Applications*, Vol. 167, p. 114116.

Guo, W., Vanhoucke, M. and Coelho, J., 2023, "A prediction model for ranking branch-and-bound procedures for the resource-constrained project scheduling problem", *European Journal of Operational Research*, Vol. 306, No. 2, pp. 579-595.

Messelis, T. and De Causmaecker, P., 2014, "An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem", *European Journal of Operational Research*, Vol. 233, No. 3, pp. 511-528.

Rice, J. R., 1976, "The algorithm selection problem", *Advances in Computers*, Vol. 15, pp. 65-118.

Teichteil-Königsbuch, F., Povéda, G., González de Garibay Barba, G., Luchterhand, T. and Thiébaux, S., 2023, "Fast and Robust Resource-Constrained Scheduling with Graph Neural Networks", *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 33, No. 1, pp. 623-633.

Verhaeghe, H., Cappart, Q., Pesant, G. and Quimper, C.-G., 2024, "Learning Precedences for Scheduling Problems with Graph Neural Networks", in Shaw, P. (ed.), *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 30:1-30:18.

Zhao, X., Song, W., Li, Q., Shi, H., Kang, Z. and Zhang, C., 2022, "A Deep Reinforcement Learning Approach for Resource-Constrained Project Scheduling", in *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1226-1234.

Modelling a satellite task schedulability constraint with neural network equations

Romain Barrault[1], Cedric Pralet[1], Gauthier Picard[1] and Eric Sawyer[2]

[1] DTIS, ONERA, Universite de Toulouse
romain.barrault, cedric.pralet, gauthier.picard,@onera.fr
[2] CNES, Toulouse
eric.sawyer@cnes.fr

Keywords: hybridizing optimization and learning, scheduling and planning, earth observation satellites.

1 Introduction

In the field of Earth Observation, a generic problem is the scheduling of satellite observation tasks. On the one hand, this problem consists in selecting the pictures to take starting from a set of requests that are submitted by end-users, and on the other hand it requires to schedule the selected observation tasks. These two types of decision lead to an extremely combinatorial problem, particularly when it comes to satellite constellations. In order to lower this complexity, the main idea in this paper consists in building a mathematical model based on an abstraction of the schedulability problem of a set of observation tasks to try to return good quality selections in a short amount of time. In that fashion, we defined an approach which allows us to represent a schedulability constraint of a set of observation tasks for a satellite thanks to a neural network. The equations that are learned and memorized within its layers are then exploited throughout a mixed integer linear programming model. This method refers to a well-known approach in the literature called *Empirical Model Learning* (EML (Lombardi *et. al.* 2017)

2 Learning a schedulability constraint

For the considered use case, a task corresponds to observing a mesh of interest on the Earth surface with the satellite on an orbit portion (cf. figure 1). Because of observation angle constraints, each of these tasks can only be accomplished during a certain time window, which is the time frame during which the satellite passes over the corresponding mesh. Additionally, transition times are required between each following observations for the satellite to have enough time to reorient its observation instrument in the right direction every time. On top of that, these transition times are *time-dependent*. Indeed, their duration depends on the satellite position when starting the transition, which depends on the date at which the observations are performed.

Given a set of meshes M to be observed, determining whether there is an order that allows all meshes to be observed while satisfying all temporal constraints is a NP-hard problem. In terms of learning, this schedulability problem can be seen as a problem of classifying such M sets. To address it, we propose to use not the raw data defining all possible sets M (Barrault *et. al.* 2025), but more targeted features. In particular, the space of meshes overflown by the satellite is represented by a grid of size $W \times H$. On this grid, we define a set of mesh blocks B. Each block $b \in B$ corresponds to a subgrid of size $W \times H_b$, and it is possible to calculate, from a set of selected meshes M, the number of meshes $nMeshes_b$ activated in each block b and the horizontal dispersion $hDisp_b$ of the activated meshes (the difference between the largest and smallest columns occupied by a mesh).

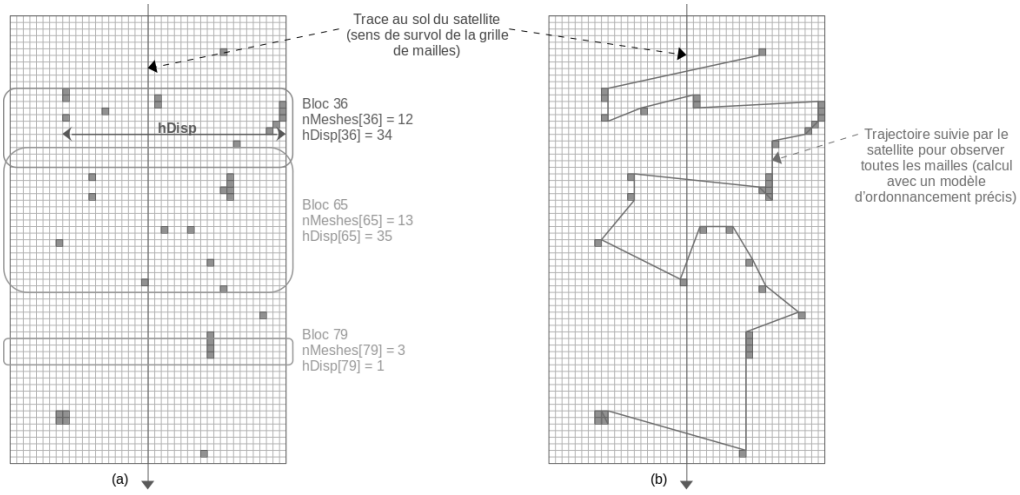


Fig. 1. (a) Surrogate model in order to determine whether a mesh selection is schedulable for the satellite; (b) model with computation of a precise observation sequence

Using a sample database, we optimize the parameters of a neural network whose inputs are $\{nMeshes_b \,|\, b \in B\} \cup \{hDisp_b \,|\, b \in B\}$ and whose output is 0 or 1. This network consists of a hidden linear layer followed by a layer involving n ReLu nodes. Other features were also tested, such as the weight of a minimum spanning tree over M.

3 Optimization model for selecting the observations

The neural network that approximates the schedulability constraint is used within an optimization model that maximizes the sum of rewards provided by observations selected from among the candidate observations. This model contains various variables:

- for each candidate mesh m, a boolean variable $select_m \in \{0, 1\}$;
- for each bloc $b \in B$ of size $W \times H_b$, variables $nMeshes_b \in \mathbb{R}^+$ et $hDisp_b \in \mathbb{R}^+$ which represent the formerly mentioned features;
- for each node k within the neural network approximating the schedulability constraint, a variable $xOutput_k$ representing the output value of node k.

Then, linear constraints allow us to formulate : (1) the link between variables $nMeshes_b$, $hDisp_b$ and initial decision variables $select_m$; (2) the relations between each input and output within the network's nodes, with a specific coding of ReLu nodes taken from existing works (Grimstad B. *et. al.* 2019); (3) the fact that the network's output should be equal to 1 for the mesh selection to be judged as schedulable. Other models using a threshold derived from the computation of the weight of a minimum spanning tree have also been explored. The selection returned by the previous MILP model is then used as the initial solution for a second optimizer, which precisely handles the computation of a sequence of observations and the corresponding temporal constraints.

4 Experiments

The classifier was trained on a dataset containing more than 28,000 instances of the problem obtained via a realistic simulator. In fact, our realistic simulator is able to compute

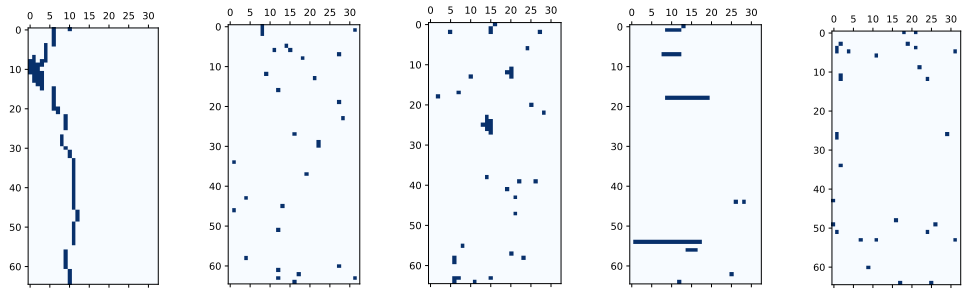


Fig. 2. From left to right : *dense*, *sparse*, *mixed*, *flat* and *borders* instance

maximal instances for our problem, which means that each computed set of observations is schedulable, and any additional mesh activation would make the observation set unschedulable. Due to problem complexity, this maximality feature is only approximated since it is obtained thanks to a greedy method, but having an exact method would allow us to add only a handful of other meshes. This set of maximal instances is then used to obtain *positive* instances for our problem, by deactivating some activated meshes, since any subset of a schedulable mesh set is also schedulable, and *negative* instances for our problem by activating some additional meshes, because of the maximality of the initial set. By choosing to activate or deactivate only a handful of meshes in most cases, we designed our train and test dataset as very close to the schedulability frontier since it is the zone at stake for optimization purpose. On top of that, all instances were separated in 5 types, which differ in observation density and position. Figure 2 displays an instance example for each of these 5 types. The *dense* instances show a high geographical density in activated meshes, both vertically and horizontally. The *sparse* instances on the other hand only have isolated activated meshes. The *mixed* instances stands in-between, with both isolated observations and small local hubs of these. The last two types are more specific : the *flat* have several horizontal observation hubs, and *borders* instances are *sparse* ones with most of their observations very close to the lateral borders. These last types are designed for the classifier to learn the difficulties for the satellite to perform lateral transitions since they are perpendicular to the satellite's movement. On the 7101 test instances, it showed an accuracy rate of 85% and performed better and better as the instances moved away from the feasibility boundary.

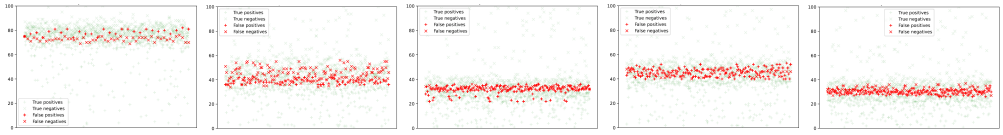


Fig. 3. Classification results for the designed classifier. On each graph, each dot corresponds to a test instance classified by the model, green if correctly classified and red otherwise. Negative instances are represented by crosses and positive instances by pluses. The x-axis is for the instance number and the y-axis is the amount of activated meshes in the instance.

Figure 3 displays the classifier's result. It showed an average error rate of 14.52%, with a higher performance *dense* instances. The schedulability frontier is clearly visible on these results as a horizontal red zone, which position on the y-axis reveals the typical size of the first unfeasible instances. The fact that this zone's position and height differ from a type to another (except between *sparse* and *borders* which are very similar) show that this dataset division is relevant.

Our results also showed that the error rate drops under 10% on average on instances that differ from the maximal instance used to create it from more than 4 meshes, and under 5% on instances differing from more than 7 meshes, which is a very satisfying characteristic.

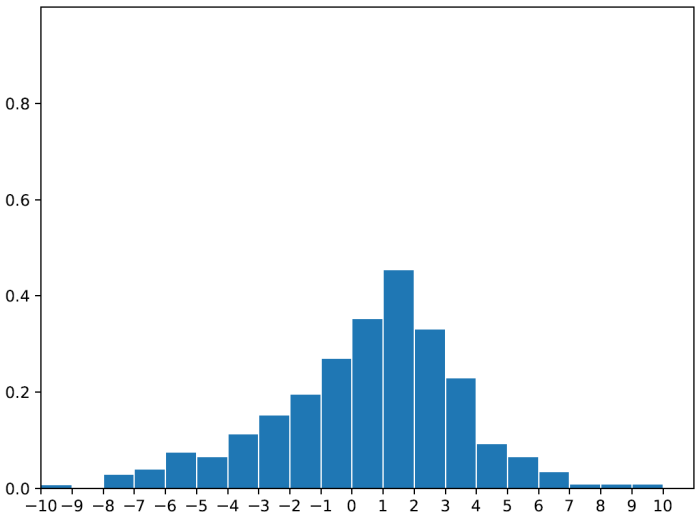


Fig. 4. Error rate of designed neural networks in function of the amount of added meshes (positive value) or substracted (negative) from its mother maximal instance, from from -10 to $+10$ meshes

Figure 4 shows the classifier's average performances. It is no surprising that the highest error rate is on maximal instances (x-axis 0) since they are the hardest to classify. It also shows that the error rate drops under 10% for instances with 5 added or substracted meshes, and 5% with 7 added or substracted meshes, which is a satisfying certainty zone. Finally, the average performance better on negative instances.

We were therefore able to implement this model within a planner using an EML (Lombardi *et. al.* 2017)-type approach, whose results were equivalent in quality to those of a good planner (Barrault *et. al.* 2025) while reducing the average time to obtain an optimal selection by a factor of 5.

References

Lombardi M., Milano M. and Bartolini A., "Empirical decision model learning", *Artificial Intelligence*, 244: 343-367.

Barrault R., Pralet C., Picard G. and Sawyer E., 2025, "Hybridizing machine learning and optimization for planning satellite observations", *CPAIOR*.

Grimstad B. and Andersson H., "ReLU networks as surrogate models in mixed-integer linear programs", *Computers & Chemical Engineering*, 131: 106580, 2019.

Compact encoding for the Job Shop Scheduling Problem for variational quantum algorithms

Quentin Perrachon[1], Eric Bourreau[2], Alexandru Olteanu[1], Marc Sevaux[1]

[1] Lab-STICC, Université Bretagne Sud, Lorient, France
{quentin.perrachon, alexandru.olteanu, marc.sevaux}@univ-ubs.fr
[2] LIRMM, CNRS, Université de Montpellier, Montpellier, France
eric.bourreau@lirmm.fr

Keywords: quantum optimization, job shop, scheduling, variational algorithm

1 Introduction

The resolution of combinatorial optimization problems using quantum computing is a rapidly expanding research field. Scheduling problems are popular among the combinatorial optimization problems due to their complexity and their practical importance. We focus here on the resolution of a Job Shop Scheduling Problem (JSSP) with n jobs and m machines (nm operations), with makespan minimization.

In the context of NISQ computers, variational quantum algorithms are currently one of the most promising quantum approaches for combinatorial optimization. They rely on short and simple circuits, suited to the high level of noise of current machines. Another major constraint of today's quantum machines lies in the limited number of available qubits, which encourages us to develop binary encoding schemes that are as compact as possible to represent our solutions.

Several quantum encoding schemes for the JSSP have already been proposed in the literature. QUBO formulations make it possible to express the JSSP in binary form by integrating all constraints into a penalized quadratic function. The most popular QUBO formulation is based on time-indexing and requires nmT binary variables, where T represents the discretized time horizon (Aggoune and Deleplanque 2023). A second approach, suggested in two different papers, uses Bierwirth vectors and mixed-radix ranking. The total number of valid Bierwirth sequences for a JSSP with n jobs and m machines is $(nm)!/m!^n$, which can be rank-encoded using mixed-radix ranking. The required number of qubits is therefore $\lceil \log_2((nm)!/m!^n) \rceil$ (Bourreau et al. 2024, Schmid et al. 2024). This encoding leads to a surjective mapping over feasible solutions, with many encodings corresponding to a single solution.

2 Variational quantum algorithm

Our solution scheme falls within the framework of a variational quantum algorithm or VQE (*Variational Quantum Eigensolver*) (Grange, Poss and Bourreau 2023). VQE is a hybrid algorithm using both a classical computer and a quantum computer.

It relies on the preparation of a parameterized quantum circuit, called an *ansatz*, whose parameters are optimized by an iterative classical meta-algorithm. The parameterized quantum circuit represents a probability distribution over the solution space of a combinatorial problem, controlled by its parameters. The optimization of the parameters therefore aims at concentrating the amplitude and increasing the probabilities of low-cost solutions (for a minimization problem).

For our application, we use a very basic parameterized quantum circuit (ansatz) composed of only a single rotation R_y around the Y-axis of the Bloch sphere on each qubit

i by an angle θ_i. The quantum circuit can be written as the following tensor product, $\bigotimes_{i=1}^{q} R_{y_i}(\theta_i)$. The parameter vector $\theta \in \mathbb{R}^q$ is optimized by a gradient descent algorithm guided by measurements of the quantum circuit from which we obtain solutions and their associated costs.

3 Contributions

We propose an alternative encoding for the JSSP, based on the representation of solutions by the sequences of jobs on each machine, corresponding to m permutations of size n. There are $n!^m$ possible encodings, and each of these encodings can be decoded into a directed acyclic graph representing a solution of the JSSP. Interpreting the permutations as strict orders would lead to infeasible solutions being encoded. Instead, each permutation is interpreted as a priority list using a simple constructive deterministic heuristic. Using this decoding procedure, all feasible solutions can be obtained from at least one encoding, with a surjective mapping between the encoded space and the space of feasible solutions.

3.1 Ranking

In order to obtain a compact encoding in terms of the number of qubits, we rank all solutions and encode only the rank of a solution. This approach relies on the ability to efficiently associate a rank with each solution and similarly, to reconstruct a solution from a given rank. Importantly, this must be achieved without explicitly enumerating the $n!^m$ solutions.

We start by associating an integer rank between 1 and $n!$ with each permutation, then concatenating these m ranks in base $n!$ into a unique representation. This final representation is interpreted as an integer, thus defining a global rank between 1 and $(n!)^m$. This rank is then binary-encoded using only $\lceil \log_2 \left((n!)^m\right) \rceil$ qubits.

Multiple procedures exist to enumerate permutations in a specific order. Many of these permutation orderings can be interpreted as walks on a structured graph whose vertices are permutations and whose edges correspond to a fixed set of simple transformations. Generating permutations in a prescribed order can be reduced to constructing a Hamiltonian path in a given permutation graph, such as a permutohedron (see Figure 1).

We are able to define ranking and unranking procedures for some ordering of permutations. For lexicographical orderings, we rely on classical factoradic representations, also known as Lehmer codes. They provide a bijection between permutations of size n and integers in $\{0, \ldots, n! - 1\}$. Ranking corresponds to computing the Lehmer code of a permutation and interpreting it as a mixed-radix integer, while unranking is obtained by the inverse factoradic decoding. For Steinhaus-Johnson-Trotter (SJT) (Johnson 1963) orderings, which generate permutations through adjacent transpositions, we can define dedicated recursive ranking and unranking procedures in $O(n^2)$. These procedures exploit the self-similar structure of the SJT sequence and its reflect-and-insert construction, allowing one to compute ranks and permutations directly, without storing or traversing the full permutation list.

These ranking and unranking functions provide a surjective and computable mapping between ranks and permutations, enabling a compact and scalable encoding suitable for variational quantum algorithms.

This encoding requires significantly fewer qubits than the previously mentioned representation. For example, a scheduling problem of 3 jobs and 3 machines would require at least 27 binary variables and qubits using a QUBO formulation and 11 qubits for Bierwirth vectors. Our encoding would use only 8 qubits for the same problem size. This gap widens for larger instances, where a problem of size 5 jobs and 5 machines would require

respectively 625, 50 and 35 qubits for the QUBO representation, Bierwirth vectors and our encoding.

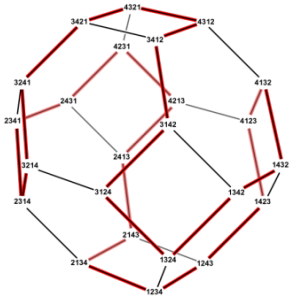


Fig. 1: Hamiltonian path in a permutation graph

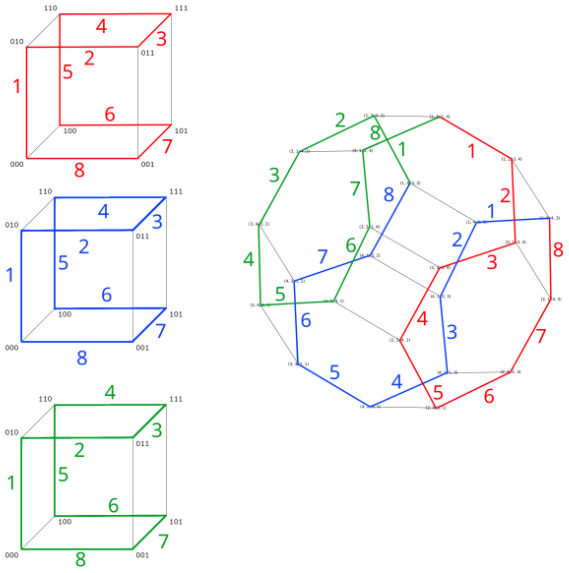


Fig. 2: Mapping between binary and combinatorial Gray code

3.2 Binary encoding

An initial naive approach would consist in encoding the rank of our solution as its standard binary representation. However, in a variational quantum algorithm, the circuit parameters act directly on the qubit register, and therefore on the binary structure of the encoding. As a result, we hypothesize that the choice of the binary representation is not neutral and may significantly impact both the landscape explored by the quantum circuit and the quality of the gradient descent meta-algorithm. The following work is done with this hypothesis in mind.

We notice that under the Steinhaus-Johnson-Trotter ordering, two consecutive ranks correspond to permutations that differ by a single adjacent transposition, a distance of one adjacent transposition in permutation space. Our goal is to preserve this property in the binary representation such that the binary encoding of two consecutive ranks in the SJT ordering differ only by a Hamming distance of 1. This is reminiscent of the notion of Gray codes. A binary Gray code is an ordering of binary strings in which two consecutive binary strings differ by exactly one bit flip.

Figure 2 shows a mapping between a Hamiltonian path on a binary structure (binary Gray code) and a Hamiltonian path on a permutation graph (combinatorial Gray code) as presented in the seminal work of Gaggini (2025). As shown in this figure, we can partition the binary encoding into subspaces. The encoding shown in this figure uses 2 bits to encode which of the 3 subspace (cube), and 3 bits to encode the 8 Gray code rank within the subspace, which corresponds to $3 \times 8 = 24 = 4!$ permutations.

By associating the rank of a solution to its position in a binary Gray code, we ensure that consecutive ranks are both adjacent in the solution space by an adjacent transposition and in the binary encoding by a bit flip.

3.3 Resolution approach

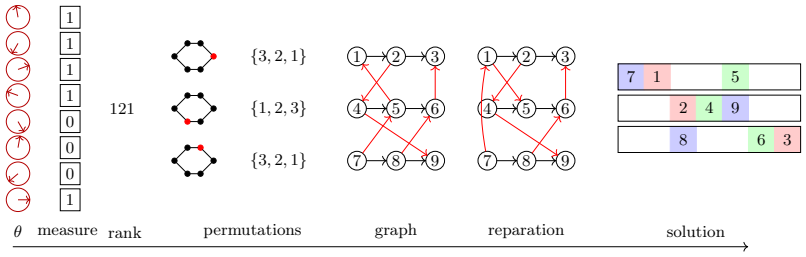


Fig. 3: Evaluation of the parameterized quantum circuit

For a given set of parameters θ, the quantum circuit prepares a superposition over binary strings, which are measured to obtain solutions using the methods described above (see Figure 3). The quality of the resulting solutions and schedules are then evaluated through a classical cost function, here the makespan, to compute the expectation value of our current parameterized circuit. This expectation value is finally used by the classical optimizer to update the circuit parameters θ using a gradient descent algorithm. This hybrid loop is repeated until convergence, allowing the variational circuit to progressively bias the probability distribution toward binary encodings associated with high-quality scheduling solutions. The largest size of JSSP (with arbitrary processing times) solved by quantum computers is 13 operations (Schmid et al. 2024) or 4×4 operations (Bourreau et al. 2024). Using the proposed encoding and this resolution approach, we have managed to find solutions to instances of size 5×5 operations, which requires 35 qubits. Initial computational results on simulators show a gap of up to 15% from the optimum in a time budget of 15 minutes. Further developments, including instance size or different permutation spaces are currently under evaluation and will be presented.

References

Aggoune R., Deleplanque S., 2023, "Solving the job shop scheduling problem: QUBO model and quantum annealing", in *Emerging Optimization Methods: From Metaheuristics to Quantum Approaches*.

Grange C., Poss M., Bourreau E., 2023, "An introduction to variational quantum algorithms for combinatorial optimization problems", *4OR*, Vol. 21, No. 3, pp. 363–403.

Schmid M., Braun S., Sollacher R., Hartman M. J., 2024, "Highly efficient encoding for job-shop scheduling problems and its application on quantum computers", *Quantum Science and Technology*, Vol. 10, No. 1, Article 015051.

Bourreau E., Fleury G., Lacomme P., 2024, "Indirect Job-Shop coding using rank: application to QAOA (IQAOA)", *arXiv preprint* arXiv:2402.18280.

Johnson S. M., 1963, "Generation of permutations by adjacent transposition", *Mathematics of Computation*, Vol. 17, No. 83, pp. 282–285.

Gaggini L., 2025, "Généralisation des techniques d'énumération pour les méthodes variationnelles quantiques", Seminal work, Université de Montpellier, `github.com/GagginiLorenzo/Representations-Operatoire`

M - Scheduling and Applications

Solving a large oral examination timetabling problem using a multidimensional knapsack MILP formulation

Cyrille Briand[1] and Jean-Pierre Belaud[2]

[1] LAAS-CNRS, University of Toulouse, France
cyrille.briand@laas.fr
[2] LGC, University of Toulouse, CNRS, Toulouse, France
jeanpierre.belaud@toulouse-inp.fr

Keywords: Examination timetabling, multidimensional knapsack , mixed integer linear programming.

1 Introduction

France's prestigious engineering schools, known as *Grandes Écoles*, train students from the third year of bachelor's degree (L3) to the second year of a master's degree (M2). Admission is determined by performance in major groups of written examinations and, if successful, oral examinations. Each group targets a distinct student profile and range of schools. The largest academic track, MP (Mathematics and Physics), historically accounts for the majority of candidates, with approximately 6,500 to 7,000 students taking the exams annually.The largest academic track, MP (Mathematics and Physics), historically accounts for the majority of candidates, with approximately 6,500 to 7,000 students sitting the exams each year. The newer track, MPI (Mathematics, Physics, and Computer Science) has a more modest contingent, around 1,000 candidates annually. While both tracks share the same written exam bank, their specific oral exams reflect the stronger emphasis on computer science for the MPI track.

The organization of oral examinations is entrusted to the Service des Concours Communs Polytechniques (SCCP), hosted by Toulouse INP University.. The SCCP is responsible for operating the oral examinations of CentraleSupélec, Mines-Ponts, and Concours Commun INP (CC INP), which provide access to a wide range of schools (approximately 80 schools with 5,700 available places). The organization is complex, as a single student may be eligible for several oral examination groups, each with its own constraints, requiring careful coordination of the individual exam timetables.

The oral examinations consist of offering each successfully admitted candidate a schedule of oral exams, i.e., a specific time slot for each exam for which they are admissible. The timeline is very tight, as schedules must be built within only a few days following the release of the admissibility results. During each scheduled slot, a candidate may take several tests (e.g., practical work, modern language oral exams). Therefore, the duration of a slot depends on the type of the exam. This is a large scale schedule problem as there are more than 7,000 candidates to accommodate over a 4-weeks time horizon. It is also complex as each candidate can be eligible for several types of oral exam, which requires to avoid overlapping slots. Furthermore, the specific modern language chosen by the candidate must be carefully considered, as not all languages are available in every slot. In addition, for a given exam type, there are several possible slots, and each is characterized by its own constraints: time length, type of slot (MP, MPI, or both), maximum number of candidates, modern languages available, maximum number of candidates for each language. Finally, since the oral examinations take place in Paris, the proposed schedule must account for the candidate's place of origin. For instance, since accommodation in Paris is expensive, the slots offered to candidates from outside Paris must avoid long idle times. For non-French

candidates, the time required to obtain a visa must be accounted for, meaning that only late slots, scheduled towards the end of the four-week period, should be proposed.

This paper tackles this peculiar Oral Examination Timetabling Problem (OETP). It first shows how this problem can be advantageously modeled as a Multidimensional Knapsack Problem (MKP), and the pros and cons of such an approach are discussed. The paper also provides an initial set of experiments on the real 2025 SCCP MP/MPI problem instance, utilizing an open-source MILP solver. These initial results demonstrate the effectiveness of the approach. The remainder of the paper is structured as follows. Section 2 presents a brief literature review of the OETP. Section 3 details our BMP-oriented modeling, which is formulated as a mixed integer linear program. Section 4 details the experimentation made. Conclusion and perspectives are drawn in Section 5.

2 Brief literature Review

The Oral Examination Timetabling Problem (OETP) can be viewed as a special case of the University Examination Timetabling Problem (UETP), which has been extensively studied over the past decades due to its inherent computational complexity. UETP is defined as the assignment of a set of exams into a limited number of time slots and rooms while satisfying a variety of hard and soft constraints, making it a classic NP-hard combinatorial optimization problem. Seminal work by (Carter et al. 1996) established the foundational methodology for UETP, primarily modeling the problem as a Graph Coloring Problem. In this paradigm, exams are represented as vertices and conflicts (shared students) as edges; the objective is to assign colors (time slots) such that no adjacent vertices share the same color. Following these early graph-theoretic formulations, the literature has expanded to include a wide variety of solution techniques. As detailed in comprehensive surveys such as the one by Qu et al. (Qu et al. 2009), approaches have evolved from constructive heuristics to sophisticated metaheuristics, including evolutionary algorithms, simulated annealing, and tabu search. Moreover, as discussed in (Tomáš Müller 2016) based on a real large-scale UETP, Constraint Programming (CP) is a powerful approach capable of conveniently handling most complex constraints, yet scalability remains a challenge for large instances.

Most existing studies address the room assignment problem using matching algorithms or network flow models, typically after the time slots have been fixed. Few studies, however, explore the simultaneous optimization of time and resources using a packing perspective, although assigning exams to rooms with heterogeneous capacities shares structural properties with the Knapsack Problem. The Multidimensional Knapsack Problem (MKP) is a well-known variation of the Knapsack Problem where items (exams) consume multiple resources (e.g., seats, invigilators, equipment) simultaneously. Given the increasing complexity of modern university requirements, where constraints are not merely about "fitting" students into rooms but about optimizing multi-attribute resources, an MKP-oriented approach could be relevant. Therefore, this paper investigates such an approach, tackling the OETP using an MKP-based Mixed Integer Linear Programming (MILP) formulation.

3 Modelling

To model the OETP, one could employ a direct method involving the creation of an assignment variable for each candidate-slot combination. While this method offers the substantial advantage of ensuring comprehensive consideration of all potential assignments, it suffers from significant drawbacks as it leads to a combinatorial explosion due to the creation of an excessive number of variables and constraints, drastically increasing the

overall model complexity. Specifically, integrating intricate operational constraints, such as non-overlapping slots or candidate origin constraints, is not trivial.

Alternatively, our approach considers only a predetermined set of existing timetables, potentially derived from historical data, and assigns one of these pre-validated schedules to each candidate. The primary benefit of this strategy is its ability to severely restrict the total number of variables and constraints, thereby simplifying the model. It also naturally supports a human-in-the-loop validation process, as schedules can be specifically designed to accommodate candidates. However, its main limitation is the requirement for a separate mechanism to dynamically construct new consistent schedules.

Formally, the OETP can be defined by the tuple $(\mathcal{C}, \mathcal{P}, \mathcal{R}, \{\mathcal{P}_i\}, \{\mathcal{P}_k\})$. $\mathcal{C}$ is the set of *candidates*, $\mathcal{P}$ denotes the set of *schedules* (or timetables), and $\mathcal{R}$ is the set of resources, each resource $k \in \mathcal{R}$ having a capacity b_k. $\mathcal{P}_i \subseteq \mathcal{P}$ is the subset of compatible schedules for candidate i. Symmetrically, $\mathcal{P}_k \subseteq \mathcal{P}$ is the subset of schedules that utilize resource k.

The objective is to maximize the total number of assigned candidates subject to capacity constraints. This problem is mathematically formulated as a Multidimensional Knapsack Problem (MKP) below. The notation is defined as follows (see Figure 1 for illustration). $y_{ij} \in \{0, 1\}$ are binary decision variables. $y_{ij} = 1$ if candidate i is assigned to schedule $j \in \mathcal{P}_i$, and 0 otherwise. The objective function (1) expresses the goal of maximizing the total number of assigned candidates. Constraints (2) are the resource capacity constraints, ensuring that the total usage of resource k by the selected schedules does not exceed the capacity b_k. Finally, constraints (3) are the assignment constraints, ensuring that each candidate i is assigned to at most one schedule $j \in \mathcal{P}_i$.

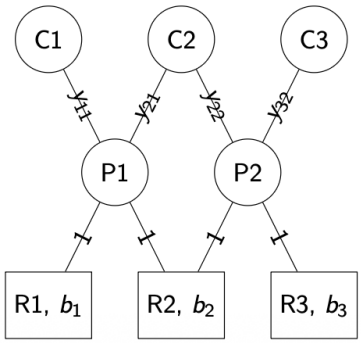


Fig. 1. Illustration of the formulation

$$\max \quad \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{P}} y_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{P}_k} y_{ij} \leq b_k, \quad \forall k \in \mathcal{R} \tag{2}$$

$$\sum_{j \in \mathcal{P}_i} y_{ij} \leq 1, \quad \forall i \in \mathcal{C} \tag{3}$$

$$y_{ij} \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{C} \times \mathcal{P} \tag{4}$$

4 Experiments

This experiment evaluates the performance of SCIP (Solving Constraint Integer Programs) on a large-scale assignment problem derived from the 2025 SCCP examination process. SCIP is a state-of-the-art framework for integer and linear optimization. It is freely available for academic use and distributed with its full source code. Its Apache 2.0 license allows unrestricted use, including in commercial contexts.

The studied MP/MPI instance involves 7,804 candidates, each characterized by a highly heterogeneous profile. The overall search space comprises 7,759 feasible schedules (or timetables). A dedicated greedy heuristic was implemented to explore all admissible assignments, derived from the 1,123,321 candidate-schedule compatibility relations. Each assignment simultaneously utilizes up to four resources from a pool of 103 available resources, each with its specific capacity. The greedy procedure produces an initial feasible solution, assigning 7,759 candidates (out of the 7,804 total). This solution utilized 606 distinct schedules. This solution was injected into SCIP as a warm start, along with the 606 schedules actually utilized by the greedy solution. A set of 100 still unsaturated plannings were also provided to SCIP, enabling the solver to focus its search on the most promising region of the solution space, under a time limit of 20 minutes. SCIP successfully computed an *optimal* solution accommodating 7,796 candidates among 7,804, thus demonstrating the benefit of combining a fast constructive heuristic with a powerful MILP solver. Assigning more candidates would require increasing the pool of available schedules (or to relax unsatisfiable constraints).

5 Conclusion and Future Work

Formulating the OETP as an MKP provides a rigorous mathematical structure, enabling the effective management of the complexity associated with capacity and resource constraints. This approach is particularly relevant in planning contexts where multiple resource dimensions compete for a set of discrete items (schedules). By framing the selection of schedules under these multiple resource constraints, the MKP model not only facilitates the exploration of the solution space but also offers theoretical guarantees for the application of exact optimization algorithms and specialized heuristics, thus validating its relevance for solving complex application cases.

To overcome the computational limitations inherent in large-scale instances, a promising direction is the adoption of a Branch-and-Price approach. This decomposition allows the Restricted Master Problem to be treated as an MKP instance, while the resulting pricing subproblem focuses on efficiently generating new columns (improving schedules) with a positive reduced cost. Once a set of interesting columns is determined, future directions could expand to integrating auxiliary objective functions, such as establishing a balanced parity across the involved academic tracks, thereby better aligning the final solution with operational needs.

References

Carter M. W., G. Laporte, D. H. Lee, 1996, “Examination timetabling: Algorithmic strategies and applications", *Journal of Operational Research Society,*, Vol. 47(3), pp. 373-383.

Müller T., 2016, “Real-life examination timetabling", *Journal of Scheduling*, Vol. 19, pp. 257-270.

Qu R., E. K. Burke, B. McCollum, I. Batyrshin, A. Meisels, 2009, “A survey of search methodologies for university timetabling problems", *Annals of Operations Research*, Vol. 174, pp. 191-219.

Logic-Based Benders Decomposition for Multi-Factory and Multi-Level Scheduling

Malick Sow[1], Guillaume Povéda[1], Florent Teichteil-Koenigsbuch[1], Christophe Louat[1] and Stéphanie Roussel[2]

[1] Airbus
el-hadji-malick.sow@airbus.com, guillaume.poveda@airbus.com, florent.teichteil-koenigsbuch@airbus.com, christophe.louat@airbus.com
[2] ONERA/DTIS, Université de Toulouse, France
stephanie.roussel@onera.fr

1 Problem Description

The aviation industry relies on distributed production networks where components are manufactured at specialized sites before being transported to Final Assembly Lines (FALs). Strategic decisions such as production allocation and operational scheduling are typically handled separately. We address this Multi-Factory Planning and Sequence-Dependent Scheduling problem jointly using a Logic-Based Benders Decomposition (LBBD) approach [Hooker, 2023]. We present the scheduling levels involved, introduce the proposed LBBD model, and compare it against a global Constraint Programming (CP) formulation.

High-Level Scheduling Problem. Production involves a set of distinct aircraft designated by MSN (Manufacturer Serial Number) where a MSN encompasses an unique identifier for a given aircraft, some specific tasks and a delivery date. A MSN is associated with Major Components Assembly (MCAs) (e.g., wings, fuselage), produced at specialized, multi-line sites (often referred to as pre-FALs). Once ready, all MCAs for a specific MSN are assembled at a FAL chosen in a set of candidate FALs available. Similarly, each MCA may be produced on one of multiple compatible assembly lines. The production process, whether on a component line or at the FAL, operates on a pulsed line system, where units move through workstations according to a fixed cycle time.

Due to asynchronous production, MCAs are often not completed at the same moment, requiring storage, for the ones that are finished sooner. This defines the high-level scheduling problem, with three associated sets of decisions: i) assignment of MCAs to production lines and MSNs to FALs, ii) sequencing of MSN and MCA on each line, iii) assigning a start production date for each product, as well as transit dates for MCAs when moving from Pre-FAL to FAL. The left part of Figure 1 illustrates the high-level scheduling problem. We consider a unique MCA, the wings, two candidate Pre-FALs and two candidate FALs for the assembled aircraft. Pre-FAL 1 is organized with 2 workstations, whereas Pre-FAL 2 has 3 workstations. For this high-level scheduling problem, decisions are as follows: allocations of all elements to candidate lines, e.g. {,

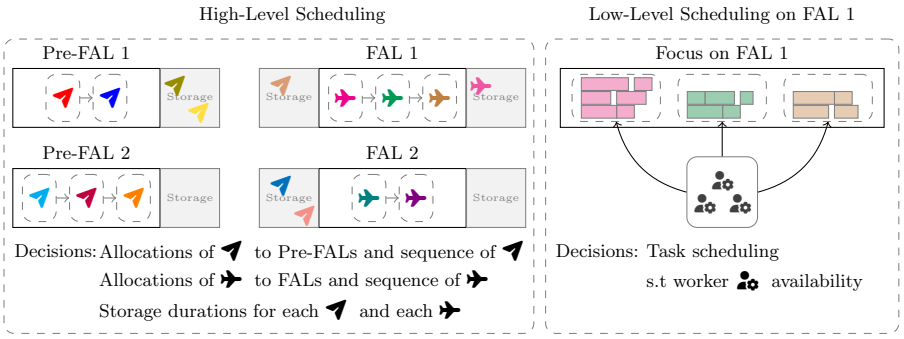


Fig. 1: Illustration of scheduling problems at each level

, , } to Pre-FAL 1, sequence of elements on the line, e.g. (,) for Pre-FAL 1, dates for transiting, e.g. entry date of in storage of FAL-2 until its production start date for the MSN in the line.

The objective is to minimize storage costs and delivery deviations. However, this high-level planning relies on aggregate capacity estimates and overlooks detailed station-level scheduling, which can lead to sequences that violate operational constraints—such as workforce availability or fixed cycle times—making the plan infeasible.

FAL Scheduling Problem. The Final Assembly Line (FAL) is a manufacturing environment comprising multiple assembly lines, each consisting of several workstations. At each station, a specific set of resource-consuming tasks depending on the MSN must be scheduled. Resources are cumulative, allowing parallel execution of tasks and therefore requiring their precise scheduling. The objective is to minimize the makespan within each station while strictly respecting total resource capacities and the maximum lead time in every station called takt. Crucially, these resources form a shared pool across all stations, deeply coupling the scheduling decisions of individual workstations. Furthermore, because task requirements are MSN-specific, the workload at any given time depends entirely on the exact sequence of MSNs currently occupying the neighboring stations. Consequently, the FAL scheduling problem is highly sequence-dependent. As this problem generalizes the classical Resource-Constrained Project Scheduling Problem (RCPSP) it is inherently NP-hard [Ganian et al., 2021].

2 Methodology

To address this hierarchical scheduling problem, we explore a Logic-Based Benders Decomposition (LBBD) model. In LBBD, a master problem determines high-level structural decisions, while subproblems check feasibility or compute refined costs; their feedback is returned as Benders cuts to guide the master

toward convergence. We adapt this framework to our setting as shown in Figure 2, decomposing the problem as follows:

- **Master Problem (Allocation & Sequencing)** – ***MP*** – The MP assigns MSNs and MCAs to production sites and determines the global production sequence. It uses estimates for storage duration, delivery deviations, and station lead times that will be updated according to cuts provided by subproblems.
- **Subproblem 1 (Timing)** – Given the fixed sequence provided by the MP, this subproblem optimizes start times to minimize storage and delivery deviations. It returns lower-bound cuts on these costs to the Master Problem.
- **Subproblem 2 (Task Scheduling)** – This subproblem validates station-level feasibility by scheduling tasks under shared worker and cycle-time constraints. It computes lower-bound cuts on station lead times. Moreover this scheduling problem can be decomposed by FAL (as there are no shared resources between FALs), allowing parallel resolution.

Subproblem 2 addresses the low-level scheduling, while the MILP Master Problem and Subproblem 1 handle high-level allocation and sequencing, Subproblem 2 uses CP to manage low-level scheduling. This hybrid approach leverages MILP's strength in sequencing and CP's efficiency in modeling storage as a cumulative resource. At each iteration, once the Master Problem produces a feasible solution, both subproblems are solved and return their respective Benders cuts, after which the master continues its search. LBBD allows to decouple the scheduling problems in every FAL: coupling constraints are handled in the Master, allowing to solve the scheduling problem in every FAL independently.

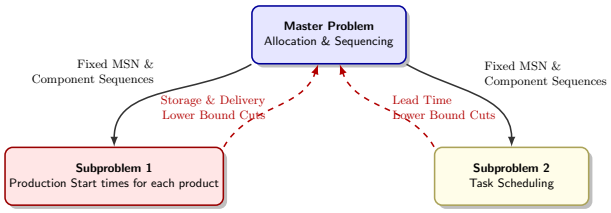
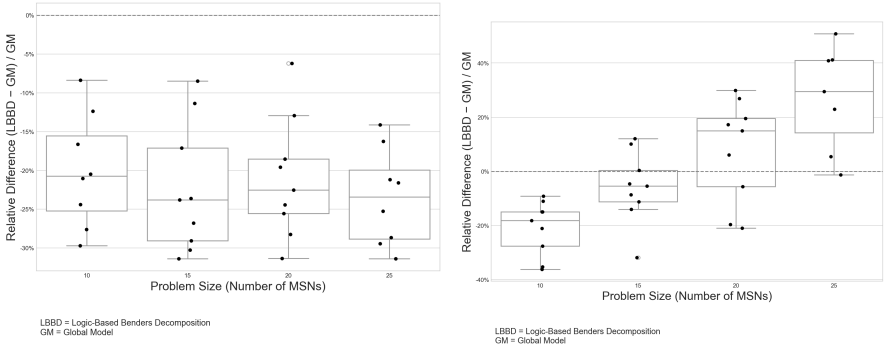


Fig. 2: Logic-Based Benders Overview

3 Results and conclusion

We compare our LBBD approach with a global CP model, using instances of 10 to 25 MSNs. CP performs well on resource-constrained problems
[Pucel and Roussel, 2024] and therefore has been selected as a baseline for our approach. As shown in Figure 3a, LBBD consistently outperforms CP for the



(a) Leadtime per station criteria (b) Earliness/tardiness + storage criteria

Fig. 3: Relative performance of CP and LBDD for several number of MSNs.

station-level lead-time objective, mainly because the decomposition enables parallel resolution of the task-scheduling subproblems.

In contrast, for high-level objectives such as earliness/tardiness and storage minimization, performance depends on instance size. LBBD is superior for small instances (10 MSNs), but its efficiency decreases for larger ones (25 MSNs), where the CP model performs better (Figure 3b). Overall, the two methods are complementary: CP handles complex global storage constraints more effectively, while LBBD is better suited to detailed, decoupled scheduling objectives.

Future Works. Because of the complementary of the two approaches explored, we believe that a hybrid approach that first utilizes the CP model to fix the primary storage and delivery targets, followed by the LBBD approach to optimize the secondary lead time per station with those constraints fixed should be investigated. Moreover, other decomposition strategy could be foreseen such as auction based multi-agent scheduling model in which each line could be seen as an agent and bid on MSNs.

References

[Ganian et al., 2021] Ganian, R., Hamm, T., and Mescoff, G. (2021). The complexity landscape of resource-constrained scheduling. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 1741–1747.

[Hooker, 2023] Hooker, J. (2023). Logic-based benders decomposition: theory and applications.

[Pucel and Roussel, 2024] Pucel, X. and Roussel, S. (2024). Constraint programming model for assembly line balancing and scheduling with walking workers and parallel stations. In *CP 2024*.

Benchmarking Trapped-Ion and Quantum Annealing Quantum Hardware on Scheduling Problems

Konrad Wojciechowski[1], Krzysztof Kurowski[1] and Jan Węglarz[2]

[1] Poznan Supercomputing and Networking Center, Poland
kwojciechowski, krzysztof.kurowski@man.poznan.pl
[2] Poznan University of Technology, Poland
jan.weglarz@cs.put.poznan.pl

Keywords: Job-shop scheduling problem, Quantum computing, Quantum annealing, Trapped-ion quantum hardware, NISQ devices

1 Introduction

Since the benchmark instances of the job-shop scheduling problem were introduced by Fisher and Thompson in (Fisher, H., Thompson, G.L. 1963), JSSP has become one of the most extensively studied problems in operations research. Although exact techniques guarantee global optimality and remain effective for small instances, their computational requirements grow rapidly with problem size. As a consequence, significant research effort has been directed towards metaheuristic frameworks and hybrid optimisation strategies that balance solution quality and computational tractability (van Hoorn, J.J. 2018). These developments provide a natural methodological context for exploring alternative optimisation paradigms within computational science, including emerging quantum algorithmic approaches. Despite rapid progress in both theoretical research and hardware development, the practical impact of quantum technologies on real-world computational problems remains an open question partially addressed in our research.

2 Problem formulation

In the job-shop scheduling problem, a set of dedicated machines is used to execute operations associated with multiple jobs. Each job consists of a predefined sequence of operations, where each operation requires a specific machine for a known processing duration. The objective is to minimize the makespan. The formulation we consider is defined as follows. There are J jobs $\mathcal{J} = \{j_1, \ldots, j_J\}$, each consisting of O_j operations $\mathcal{O}_j = \{O_{j1}, \ldots, O_{jO_j}\}$. Each operation $O_{j,k}$ has a duration time $l_{j,k}$ and must be processed on a specified machine from a set $\mathcal{M} = \{m_1, \ldots, m_M\}$.

We define binary variables that encode starting times:

$$x_{j,k,t} = \begin{cases} 1 & \text{if operation } O_{j,k} \text{ starts at time } t \\ 0 & \text{otherwise.} \end{cases}$$

In this work, the term *problem Hamiltonian* refers to the quadratic cost function used as Quadratic Unconstrained Binary Optimization (QUBO) formulations, i.e., the weighted sum of penalty terms mapped to quantum annealing hardware. The hard constraints of the problem, as well as additional optimization objectives such as the minimal makespan soft constraint and a regularization factor, are added to the cost function as a weighted sum (Kurowski, K., Wojciechowski, K., Węglarz, J. 2023).

- Single-start constraint $h_1(x)$: Each operation must start once and only once.
- Precedence constraint $h_2(x)$: Operations within jobs must respect execution order.
- Machine sharing constraint $h_3(x)$: At a given time no two operations may run on the same machine.
- Minimal makespan term $h_4(x)$: Promotes low-makespan schedules.

All these constraints are encoded in the coefficients of the Q matrix:

$$Q = \sum_{i=1}^{4} w_i \cdot h_i(x). \tag{1}$$

3 Discrete Quadratic Model Formulation

With the D-Wave Advantage annealer we explored Discrete Quadratic Models (DQMs), where variables are internally represented using one-hot binary encodings. The hybrid solver integrates classical components, including decomposition heuristics and tabu search, therefore the obtained results reflect hybrid classical and quantum optimisation rather than a purely quantum process. The DQM formulation follows the same modelling principles as the QUBO representation introduced above. In particular, the objective contains penalty terms corresponding to the single-start, precedence, and machine-sharing constraints. The single-start term ensures that each operation is assigned exactly one starting time. The precedence term enforces the technological order of operations within each job, i.e., a successor operation cannot begin before its predecessor has been completed. The machine-sharing term penalises overlapping assignments of operations that require the same machine as discussed in (Kurowski, K., Węglarz, J., et al. 2020).

4 Quantum Approximate Optimization Algorithm (QAOA) Formulation

The Quantum Approximate Optimization Algorithm (QAOA) is a variational optimisation framework designed for combinatorial optimisation problems expressed in a quadratic binary form. From an operations research perspective, QAOA can be interpreted as a parameterised search procedure operating on a probabilistic representation of feasible schedules, where the objective function

is encoded as a quadratic cost operator as discussed in (Kurowski, K., Wojciechowski, K., Węglarz, J. 2023). The corresponding cost Hamiltonian represents the encoded objective function together with feasibility penalties. QAOA constructs a parameterised quantum state through alternating applications of a cost operator and a mixing operator, starting from an initial uniform superposition state. The variational parameters are then optimised by a classical outer loop. Due to current quantum hardware limitations, only shallow circuit depths and small scheduling instances are considered. Consequently, QAOA is used here primarily as a modelling and evaluation framework rather than as a method intended to demonstrate computational advantage. Nevertheless, the formulation provides a unified interface between the QUBO-based representation used for quantum annealing and the gate-based trapped-ion platform studied in this work.

5 Experimental studies

The proposed benchmarking methodology applied in expermental studies is based on formulations of the job-shop scheduling problem for selected quantum computing platforms. Using different encoding strategies, we implement and analyse quantum algorithms in the context of computational science applications, evaluating their behaviour on representative small-scale scheduling instances with respect to fidelity and time-to-solution (TTS).

Table 1. Probability P_S and TTS measured on two different quantum computers, quantum annealer (D-Wave Advantage) and trapped-ion (EuroHPC PIAST-Q). Because the hardware paradigms differ significantly, the values should be interpreted qualitatively rather than as a direct performance ranking.

Quantum computer (instance J x M)	P_S	TTS [s]
D-Wave Advantage (6 x 6)	$2.56 \cdot 10^{-1}$	$8.41 \cdot 10^{-1}$
EuroHPC PIAST-Q (3 x 3)	$6.93 \cdot 10^{-2}$	3.20

The objective is not to establish a direct performance ranking between hardware architectures, but rather to analyse modelling choices, algorithmic workflows, and practical limitations arising from contemporary quantum computing systems. Because the JSSP problem is strongly NP-hard and difficult to solve (van Hoorn, J.J. 2018), only small instances are currently feasible on NISQ hardware. The goal of the experiments is therefore exploratory benchmarking rather than demonstrating computational advantage.

In our experiments, we used the hybrid D-Wave Advantage DQM solver and obtained all global minima during 500 shots for the Fisher and Thompson (FT06) benchmark. The FT06 instance has multiple equivalent optimal schedules with makespan 55, and therefore many optimal solutions correspond to

distinct schedules with identical objective value. For the new EuroHPC PIAST-Q trapped-ion quantum computer, we applied QAOA to a simplified JSSP instance (*3 jobs and 3 machines*), as reaching the global optimum for larger JSSP instances was not feasible. Probability P_S denotes the empirical probability of measuring an optimal solution, while TTS represents the expected runtime required to obtain at least one optimal solution with confidence level 0.99, see Table 1.

6 Conclusions

This study investigates modelling aspects of the job-shop scheduling problem within two emerging and relevant quantum-oriented optimisation frameworks tested on quantum annealing and trapped-ion quantum computers. However, the obtained results should be interpreted primarily from the perspective of formulation design and implementation feasibility rather than computational superiority. The hybrid D-Wave approach enabled experiments on a larger instance, but the observed performance reflects the contribution of integrated classical optimisation procedures and therefore does not constitute a direct comparison of quantum hardware capabilities. Given the differences in modeling assumptions, solver architecture, and hybrid classical–quantum workflows, the presented experiments should be viewed as a preliminary methodological study rather than as a direct performance comparison between EuroHPC PIAST-Q and D-Wave systems.

Acknowledgements

The authors thank the QEC4QEA project for application support and testing. Part of this work was performed using the EuroHPC JU trapped-ion PIAST-Q quantum computer.

References

Fisher, H., Thompson, G.L., 1963, 'Probabilistic learning combinations of local job-shop scheduling rules," *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, NJ, pp. 225–251.

van Hoorn, J.J., 2018, 'The current state of bounds on benchmark instances of the job shop scheduling problem," *Journal of Scheduling*, Vol. 21, pp. 127–128.

Kurowski, K., Węglarz, J., et al., 2020, 'Hybrid Quantum Annealing Heuristic Method for Solving Job Shop Scheduling Problem," In: Krzhizhanovskaya, V. et al., *Computational Science – ICCS 2020*, Lecture Notes in Computer Science, Vol. 12142, Springer.

Kurowski, K., Pecyna, T., Slysz, M., Różycki, R., Waligóra, G., Węglarz, J., 2023, 'Application of quantum approximate optimization algorithm to the job shop scheduling problem," *European Journal of Operational Research*, Vol. 310(2), pp. 518–528.

Integrating product scope and project scope. Lessons learned from teaching project management to startup companies

Denis Solan[a], Avraham Shtub[a], Moshe Weiler[b], Fernando Gómez-Baquero[c]
[a]Faculty of Data and Decisions Science Technion – Israel Institute of Technology, Haifa, Israel
denis.solan@campus.technion.ac.il shtub@technion.ac.il
[b]Gordon Center for Systems Engineering Technion – Israel Institute of Technology, Haifa, Israel
m_weiler@zahav.net.il
[c]Jacobs Technion-Cornell Institute, Cornell University, 2 West Loop Road, New York, NY 10044
fernando@cornell.edu

Keywords: Product scope, Project scope Project management education and training, New product development, Project management simulators

1. Startup training – the problem

Entrepreneurs develop products and services that provide value to customers. The development success depends on the knowledge and skills from different disciplines, but entrepreneurs are frequently experts in one field and lack knowledge in others. Very little attention is paid to integrated training in Systems Engineering and project management in entrepreneurship courses (Mwasalwiba, 2010).

There are many methodologies for addressing specific engineering challenges; however, there is a gap in understanding which engineering methodologies are suitable for start-ups. We address these gaps by developing a new approach to entrepreneurship education and training focusing on new product development projects. An integrated systems engineering and project management workshop prototype was tested focusing on the needs of entrepreneurs in start-up incubators. This new workshop was tested in the Runway incubator of Cornell Tech University in the United States. The training relied on tools and techniques of new product development, project management and training simulators.

This paper focuses on the selection and integration of tools and methods that are suitable for entrepreneurs developing new products. The insights can support entrepreneurship education and training programs and improve new product development processes.

2. Background

2.1. Entrepreneurship

Entrepreneurship the process of discovering and exploiting profitable opportunities. In many start-ups, market needs are not well understood. Due to the uncertainty and the fact that some entrepreneurs lack organizational, managerial, and legal skills.

Entrepreneurs occasionally develop new products without a clear understanding who needs and who will buy these products. As a result, some entrepreneurs fail to understand the suitability of their solution to the problem they try to solve. In successful organizations, the development process typically involves iteration and feedback from potential stakeholders to ensure a valuable system that satisfies the customer's needs and expectations.

There are different methodologies for new product development. For example, "design thinking" intends to understand customer needs and expectations in order to discover market opportunities, while "agile" development is aimed at incremental improvements (Thesing et al., 2021).

2.2. New Product Development Projects

New product development projects attempt to provide value while operating in an uncertain environment (Marzi et al., 2020). Such projects require extensive development and testing efforts and sometimes the construction of prototypes along the way. In some failed projects the underlying causes of failure can be attributed to the pre-development phase of new product development projects. This project front end phase includes generating and evaluating ideas, developing a system

concept, and setting priorities. Reaching the right solution depends on engineering and or scientific knowledge and technical expertise, as well as on design thinking, problem solving, and project management (Schuelke-Leech, 2021).

2.3. Systems Engineering

"Systems Engineering (SE) is an interdisciplinary approach that enable the realization of successful systems. It focuses on a holistic and parallel understanding of the needs and expectations of stakeholders; Examining opportunities; Requirements documentation; and synthesis, validation and development of solutions considering the complete problem, from the research of the system concept to the scrapping of the system. (SEBoK 2022).

Development requires an understanding of the principles of systems engineering. However, Systems Engineering tools and methods are not always used by small organizations and startups because they are difficult to understand and to implement. The International Council for Systems Engineering INCOSE tries to establish tailored practical guidance to improve product development through proven Systems Engineering methodology. It is important to note that sometimes, a product is not considered complex enough to warrant a complete Systems Engineering undertaking (SEBoK 2022,).

Startups may have little experience in using development processes; therefore, tools and methods are opportunistically selected to provide value under constraints. Startups can benefit from Systems Engineering methods; however, few studies deal with the application of these methods for early-stage startups. For example, Klotins et al. (2019) studied 88 start-up companies and found that the main cause of project failure is due to requirements engineering deficiencies. When there is no user involvement early on, it is impossible to correctly define requirements for potential solutions (SEBoK 20222). Therefore, agile techniques are used for early trial of functions and incremental development of requirements (Ebert & Kirschke-Biller, 2021).

Startups usually prefer simple tools that require little training. There are many methods designed to deal with specific engineering challenges. However, there is a need to help entrepreneurs focus on adapting approved methodologies and validating methodologies specific to startups.

2.4. Project management

The goal of the project management process is to systematically plan and execute project tasks to meet project objectives. Studies that have examined startup management methodologies have focused primarily on business and financial planning, but startup project management is rarely examined. This is because many startups perceive traditional project management as a "waste of time" their assumption is that uncertainty makes the long-term scheduling effort pointless. However, Entrepreneurs seeking funding from investors are required to present a business plan. The business plan typically includes a Gantt chart with project activities and milestones, which is based on traditional project management principles. Advanced methods of project planning are rarely implemented due to their complexity and lack of managerial knowledge.

Project management of start-ups is typically supported by maintaining short, informal internal milestones. This approach is especially common in software and information technology development projects, where scope and requirements change rapidly during the project lifecycle (Azenha et al. 2021).

2.5. Enterprise Development Programs

Incubators have been established around the world since 1959 to encourage the creation of new businesses by offering services to entrepreneurs and start-ups, including training sessions. Some incubators provide workshops that help shape the system's ideas. Since most entrepreneurs in an incubator are starting a company for the first time, the survival and growth of their new businesses depend on these services. While incubator business development training has received a lot of attention, most training is lacking in new product development projects. Thus, it is important to identify appropriate requirements engineering techniques for start-ups and develop appropriate training methods for these techniques.

3. The solution

3.1. Steps taken

The first step in implementing the proposed solution was to define the basic knowledge and the skills that entrepreneurs need to develop a new product. The second step was to develop and test a methodology for the integrated training of this knowledge focusing on simple tools and methods that can be applied easily. The third step was to run a workshop in the Runway incubator integrated with a simulator for planning and executing projects.

The workshop focused on very basic questions such as:

a) Do you understand the uniqueness and superiority of the proposed system?
b) What tools and techniques can be used to improve product design?
c) Do you understand the project plan (schedule, cost, and resource management)?
d) What tools and techniques can be used to improve the project plan?

3.2. Product Design Tools and Techniques

Statement of Intent

New projects typically begin with an exploratory research phase that involves business analysis and understanding the needs and expectations of stakeholders (SEBoK 2022). Since in some cases early-stage entrepreneurs fail to understand these needs and expectations they fail later. To succeed, startups must first research the problem and then validate the proposed solution. Few startups implement market-driven requirements engineering methods to discover and validate ideas before releasing their new product. The market-driven demand estimates are primarily based on entrepreneurs' assumptions and interpretations of the market.

A statement of intent outlines what the system will do and what its purpose is (SEBoK 2022). Structured formulation of the system idea helps identify high-level requirements, which are broken down into more specific functional and qualitative requirements.

FAST - Function Analysis System Technique

Unclear requirements can cause the system to be over-engineered or under-engineered. Because the requirements aren't known, a possible approach starts with creating a sustainable Minimum Viable Product (MVP) to test the concept. However many Entrepreneurs prefer simple tools for writing a list of features rather than outlining requirements.

FAST diagrams are useful in the early stages of product design to describe functions (SEBoK 2022). The technique contributes to the design of the product by encouraging thinking about functions rather than physical components. The diagrams distinguish between basic functions that are essential to the product and secondary functions that may be unnecessary and do not add much value. This function-based approach helps teams better understand the underlying reasons behind the product.

3.3. Project Management Tools and Techniques

Project Management Simulators

Many fields, including new product development, use simulations for training. Simulation-based training relies on learning by doing. We used two simulators in the workshop. The first simulator is designed for traditional (Waterfall) project management training, where project scope, cost, time, and quality are planned in advance. The second simulator is designed for Agile project management training, which relies on short-term planning and constant re-planning of the work.

Project Team Builder Training PTB, Simulator enables traditional training in new product development in a dynamic environment (Solan & Shtub, 2023). Entrepreneurs can simulate any NPD project, analyse compromises, and take risks without suffering the consequences they will face in the real world. The PTB simulates case studies called scenarios. Entrepreneurs can learn to

efficiently plan, execute, supervise, and control possible project plans using the simulator's ability to simulate various scenarios subject to schedule, budget, and resource constraints.

The Agile Management Simulator enables training in agile project management in a dynamic environment. Entrepreneurs can simulate agile projects and take risks without worrying about real-world consequences. The AGASIM simulates case studies called user stories. Entrepreneurs can learn to plan, execute, monitor, and control project sprints by planning product backlog items, managing changes, and updating product backlogs.

3.4. Lessons learned

The methodology forms the basis of a course that combines project management and systems engineering tools while integrating project scope and product scope. A similar course developed at the Technion about ten years ago focused only on the project scope of new product development. This course is available on the Coursera platform (https://www.coursera.org/learn/new-product-development). This course introduces the importance of the voice of the customer (Cooper, 2019), a Kano model developed by Noriaki Kano in the 1980s to understand how different features of a system or service affect customer satisfaction, the AHP (Analytic Hierarchy Process) technique developed by Thomas L. Sathi in the 1970s to quantify the relative importance of product requirements. In addition, the integrates the PTB methodology simulator that demonstrates the planning, analysis, and execution processes of the project.

The combination of product scope and project scope within the framework of the proposed methodology was tested in a workshop at Runway the technological incubator of Cornell Tech. The participants feedback was very positive and provided evidence that a methodology for managing new product development projects is needed and could help startup companies by reducing the probability of mistakes made early on in the project life cycle.

References

Azenha, F. C., Reis, D. A., & Fleury, A. L. (2021). The role and characteristics of hybrid approaches to project management in the development of technology-based products and services. Project Management Journal, 52(1), 90–110.

Cooper, R. G. (2019). The drivers of success in new-product development. Industrial Marketing Management, 76, 36–47.

Ebert, C., & Kirschke-Biller, F. (2021). Agile systems engineering. IEEE Software, July/August, 7–15.

Guide to the Systems Engineering Body of Knowledge SEBoK v. 2.6, May 2022. INCOSE, the IEEE Systems Council, and Stevens Institute of Technology.

Marzi, G., Ciampi, F., Dalli, D., & Dabic, M. (2020). New product development during the last ten years: The ongoing debate and future avenues. IEEE Transactions on Engineering Management, 68(1), 330–344.

Mwasalwiba, E. S. (2010). Entrepreneurship education: a review of its objectives, teaching methods, and impact indicators. Education + Training, 52(1), 20–47.

Solan, D., & Shtub, A. (2023). Development and implementation of a new product development course combining experiential learning, simulation, and a flipped classroom in remote learning. The International Journal of Management Education, 21, 100787.

Thesing, T., Feldmann, C., & Burchardt, M. (2021). Agile versus Waterfall project management: Decision model for selecting the appropriate approach to a project. Procedia Computer Science, 181, 746–756.

N - Machine Scheduling #2

Scheduling of Splittable Tasks with Uniform Setup Time on Parallel Machines

Aurélien Mombelli, Matthieu Py, Laurent Deroussi, Nathalie Grangeon, Sylvie Norre

Université Clermont Auvergne, CNRS, Clermont Auvergne INP, Mines Saint-Etienne, LIMOS, 63000 Clermont-Ferrand, France
prenom.nom@uca.fr

Keywords: Parallel Machines, Setup Time, Splittable Tasks.

1 Introduction

We are studying a scheduling problem where tasks can be split and processed by multiple machines. A setup time is required on a machine to switch from one task to another. During this setup, a machine cannot process or prepare another task. Setup times are independent of the task, the machine, and the sequence. Tasks have a minimum processing time, and machines must be in use at all times within a fixed horizon. The goal is to minimize the number of setups required. Figure 1 shows an example of scheduling on four machines, a five-hour horizon, and the following minimum durations: $T_1 \geq 6$, $T_2 \geq 2$, $T_3 \geq 2$, $T_4 \geq 1$, $T_5 \geq 1$, $T_6 \geq 1$, $T_7 \geq 1$.

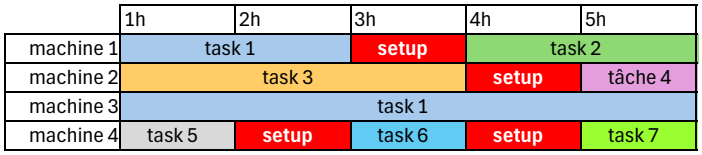


Fig. 1. Example of scheduling seven tasks on four machines over a five-hour time horizon. Tasks 1 and 3 take one hour longer than their minimum duration.

This problem is very similar to the one studied by Schalekamp et al., whose objective is the minimization of the makespan. Their study describes properties of optimal solutions, demonstrates the NP-hard complexity of the problem, and proposes a polynomial-time approximation scheme.

Our problem differs slightly in two points: our tasks have a non-fixed minimum duration and our objective is to minimize the number of settings.

2 Problem modeling

We chose to discretize the time horizon (24 hours) into one-hour slots. During each slot, the machine either performs a task or is in a setup phase. Each setup phase lasts one hour.

Using the following notation:

- M: the set of machines, $i \in M$ a machine
- N: the set of tasks, $j \in N$ a task
- $N' = N \bigcup \{0\}$: the set of tasks and settings, $j \in N'$ a task or a setting
- $(T_j)_{j \in N}$: the set of minimum assignment times per task (T_j is the minimum time in hours for task j)
- C: the set of 24 one-hour slots.

- $(X_{i,j,t} \in \{0,1\})_{\substack{i \in M \\ j \in N' \\ t \in C}}$: the set of decision variables assigning the task j on the machine i at the slot t.

The problem can therefore be written as follows:

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{\substack{i \in M \\ t \in C}} X_{i,0,t} && && (P.*) \\
\textbf{subject to} \quad & \sum_{j \in N'} X_{i,j,t} = 1 && \forall \underset{t \in C}{i \in M} && (P.1) \\
& X_{i,j,t} \geq X_{i,j,t+1} + X_{i,0,t+1} && \forall \underset{t \in C \setminus \{24\}}{\substack{i \in M \\ j \in N}} && (P.2) \\
& X_{i,0,1} = 0 && \forall i \in M && (P.3) \\
& \sum_{\substack{i \in M \\ t \in C}} X_{i,j,t} \geq T_j && \forall j \in N && (P.4)
\end{aligned}$$

The objective (P.*) is to minimize the number of settings across all machines and time slots. (P.1) requires that one and only one task or setting be selected on each machine and each time slot. (P.2) indicates that a task (other than a setting) is followed by itself or another setting. (P.3) prevents a setting from being selected on the first time slot. (P.4) requires that each task be allocated enough time to meet its minimum duration.

To help in finding a solution, certain properties of the solutions are sought:

Proposition 1: *Given a task j such that $T_j \geq 24$, there exists an optimal solution in which 24 slots are allocated on a machine to this task.*

In other words, for every task j such that $T_j \geq 24$, there exists a solution in which all the slots of a machine are allocated to task j. It is then possible to remove that machine and 24 units from T_j, thus simplifying the instance.

Proof: Assume without loss of generality that $T_j < 48$, otherwise repeat the following steps as many times as necessary. Since $24 \leq T_j < 48$, assume an optimal solution such that task j appears on two different machines. The two sets of slots E_j^1 and E_j^2 are such that $E_j^1 \geq 24/2$ and $E_j^2 \geq 24 - E_j^1$. Therefore, there exists a simple transformation (see Figure 2) that swaps all the remaining slots from the first machine with an equal number of slots at the end of E_j^2. By doing so, the number of setups either remains unchanged (and the new solution is therefore also optimal) or decreases by 1, which would contradict the optimality assumption. ■

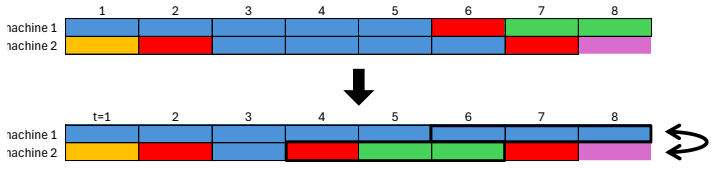


Fig. 2. A simple transformation that transform any optimal solution that respect the hypothese to an optimal solution respecting Proposition 1.

Thus, we will preprocess all our instances to search only for optimal solutions that respect proposition 1. It is then possible to reduce the instance size by removing groups of 24-hour assignments. From here on, we assume that $\forall j, T_j < 24$.

Proposition 2: *In an optimal solution, if a task is assigned to a machine, then the set of slots for that task on that machine is contiguous.*

Proof: Let's assume an optimal solution, where t_d^1 and t_f^1 are respectively the start and end slots of the first contiguous set assigned to task j, and t_d^2 and t_f^2 are respectively the start and end slots of the second contiguous set assigned to the same task j. There are three possibilities (see Figure 3):

Case 1: If $t_f^1 + 1 = t_d^2$, then the two sets are contiguous.

Case 2: If $t_f^1 + 1 = t_d^2 - 1$, then the slot separating these two sets is a setup according to the constraint $(P.2)$. It suffices to replace it with task j to obtain a better solution, which contradicts the optimality assumption.

Case 3: Otherwise, there are at least two setups (at $t_f^1 + 1$ and at $t_d^2 - 1$). Let us then consider the following exchange operator: with $D_2 = t_f^2 - t_d^2 + 1$ and $L = t_d^2 - t_f^1 - 1$

$$f(X_{i,j,t}) \begin{cases} X_{i,j,t} & \forall i,j,t \in \{0,\cdots,t_f^1\} \\ X_{i,j,t+L} & \forall i,j,t \in \{t_f^1+1,\cdot,t_f^1+D_2\} \\ X_{i,j,t-D_2} & \forall i,j,t \in \{t_f^1+D_2+1,\cdots,t_f^1+D_2+L\} \\ X_{i,j,t} & \forall i,j,t \in \{t_f^1+D_2+L+1,\cdots,24\} \end{cases}$$

The operator allows the creation of a new valid solution (the setups between the two sets of task j ensure this). However, the setups around the second set of task j end up side-by-side after the exchange operator. This contradicts the optimality assumption. ■

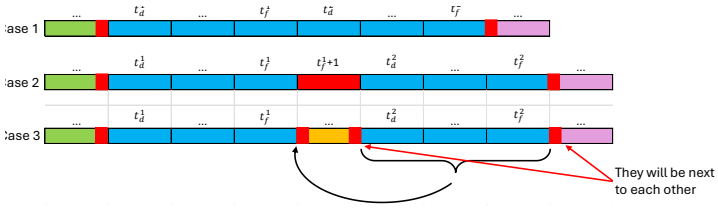


Fig. 3. If two parts of a task are assigned to the same machine, there can be only three possible configurations.

3 Upgraded model

While Proposition 1 helps us to reduce the instances size, the proposition 2 can be formulated as valid constraints and added to the model.

With $(Y_{i,j,t} \in \{0,1\})_{\substack{i\in M \\ j\in N' \\ t\in C}}$: the set of variables representing the fact that task j is starting on the machine i at the slot t.

Then, we can add the following constraints:

$$Y_{i,j,1} = X_{i,j,1} \qquad \forall \begin{smallmatrix} i\in M \\ j\in N \end{smallmatrix} \qquad (P.5)$$

$$Y_{i,j,t} \geq X_{i,j,t} - X_{i,j,t-1} \quad \forall \begin{smallmatrix} i\in M \\ j\in N \\ t\in C\setminus\{1\} \end{smallmatrix} \qquad (P.6)$$

$$\sum_{t\in C} Y_{i,j,t} \leq 1 \qquad \forall \begin{smallmatrix} i\in M \\ j\in N \end{smallmatrix} \qquad (P.7)$$

(P.5) requires that the task assigned at time 1 is started (in Y sense). (P.6) forces $Y_{i,j,t}$ to be equal to 1 if task j is assigned to machine i at time t but not at time $t-1$. Finally, (P.7) keeps Y at most to 1 for each task and machines over the time horizon.

4 Experimentations

We used CPLEX and C++17 on random instances based on an industrial real problem with the following characteristics:

- $M \in [5, 20]$
- $N \in [1.5M, 3M]$
- $T_j = \frac{mean}{2} + \lfloor \frac{j}{mean} \rfloor$ where $mean = \lfloor 0.75 * (\frac{24M-N}{N}) \rfloor$

Those characteristics were chosen because the objective value of the optimal solution is very close (equal in most cases) to $N - M$.

As each task that isn't assigned to a machine in the first time slot will require at least one setup (if not split), we know that the objective is at least $N - M$. To help the solver, this lower bound is added to the formulation.

In the following tables, UB denotes the best integer solution's objective value, LB is the best dual value and $Time$ is the CPU time to compute the solution. We stopped at 300 seconds because more computation will not yield better results in scheduling problems in general, and proved to be the case in our problem too.

The first table shows the results of our problem with constraints (P.1) to (P.4) and the second table shows the results with all the constraints.

M	N	UB	LB	$Time$
5	8	3	3	0.1
5	10	5	5	0.51
5	13	8	8	1.5
5	15	10	10	0.02
10	15	5	5	3.64
10	20	10	10	20.5
10	25	15	15	139.7
10	30	20	20	94.2
15	23	9	8	300.0
15	30	18	15	300.0
15	38	29	23	300.0
15	45	32	30	300.0
20	30	13	10	300.0
20	40	39	20	300.0
20	50	38	30	300.0
20	60	56	40	300.0

Table 1. Results of 16 instances with constraints (P.1) to (P.4).

M	N	UB	LB	$Time$
5	8	3	3	0.6
5	10	5	5	0.67
5	13	8	8	0.05
5	15	10	10	0.05
10	15	5	5	1.6
10	20	10	10	12.31
10	25	15	15	16.75
10	30	20	20	22.42
15	23	8	8	40.01
15	30	15	15	54.02
15	38	23	23	75.5
15	45	30	30	179.7
20	30	13	10	300
20	40	31	20	300
20	50	37	30	300
20	60	54	40	300

Table 2. Results of the upgraded model on the same 16 instances.

5 Conclusions et perspectives

This problem is close in its definition of Schalekamp and al. problem, yet the optimal solution's characteristics are very different. We showed that adding those characteristics to the model yield better solutions than without, and is promising for the bigger problem we want to study: some couple of tasks should be done at the same time on two different machines. What will be the characteristics of this new problem's optimal solution? Can part of them be inferred from the problem we studied in this paper? Also, as the industrial project is much larger, we will create approximation schemes to solve them.

References

Schalekamp F., Sitters R., van Der Ster S., Stougie L., Verdugo V., van Zuylen A., 2015, "Split scheduling with uniform setup times", *Journal of scheduling*, Vol. 18(2), pp. 119-129.

An efficient insertion heuristic for shop scheduling problems

Karim Tamssaouet

Department of Accounting and Operations Management, BI Norwegian Business School, 0484 Oslo, Norway
karim.tamssaouet@bi.no

Keywords: shop scheduling, construction heuristic, insertion techniques

1 Introduction

Shop scheduling problems constitute one of the most studied classes in operations research because they capture in different contexts the essence of allocating scarce processing resources to interdependent tasks over time. Classical variants (job shop, flow shop, open shop) have quite different structures, and real applications add a large diversity of constraints (e.g., release dates, sequence-dependent setups, non-linear routes, or flexible multi-resource operations (Pinedo, 2016; Dauzère-Pérès et al., 2024)). Most objectives are regular, i.e., non-decreasing in job completion times; makespan is the most studied, but min-sum criteria such as total weighted tardiness are often more relevant in practice.

Since these problems are NP-hard, good quality schedules are usually obtained with heuristics. Construction heuristics build a solution greedily from scratch, while improvement methods (local search, tabu, evolutionary algorithms) start from one or several such solutions and refine them. Despite the success of improvement methods, comparatively less attention has been paid to making construction heuristics both efficient and broadly applicable.

In practice, offline construction heuristics or real-time scheduling are mostly based on dispatching/priority rules because they are simple, transparent and fast (Artigues et al., 2005). However, insertion-based heuristics (e.g., NEH Nawaz et al. (1983)) are known to produce superior schedules, and similar ideas exist for open and job shops (Bräsel et al., 1993; Werner and Winkler, 1995). The limited adoption of these methods stems largely from their higher computational cost and the lack of generic insertion schemes. This work addresses these challenges by proposing an insertion-based construction heuristic targeting the broad class of problems representable as a directed acyclic graph (DAG) with non-negative arc weights and regular objectives. By building on Tamssaouet and Dauzère-Pérès (2023) and generalizing insertion techniques typically reserved for improvement heuristics (Mastrolilli and Gambardella, 2000; Artigues and Roubellat, 2002; Kis, 2003), this approach enables a generic yet efficient construction method. From a practical standpoint, this unlocks dual value: the heuristic serves as a superior standalone alternative to dispatching rules in time-critical contexts, or as a mechanism to provide high-quality initial solutions for warm-starting exact approaches and metaheuristics.

2 Problem modeling

The problem is specified in a general way so that many shop scheduling environments appear as special cases. We are given a set of jobs J to be processed on a set of resources R. As in most machine scheduling settings, we assume (a) each resource can handle at most one operation at a time (disjunctive constraint) and (b) processing is non-preemptive. Each job $j \in J$ is described by a route graph, i.e., a simple directed acyclic graph $G_j = (V_j, A_j)$. The nodes $v \in V_j$ represent the operations of job j and each node is associated with a set of modes $\mathcal{M}_v$; a mode $M_k \in \mathcal{M}_v$ specifies the subset of resources $M_k \subseteq R$ that must process that operation simultaneously. The arcs A_j encode the technological or precedence constraints between operations of the same job. Jobs may additionally have attributes such as due dates or weights, which are required to evaluate the objective function.

To initialize an empty schedule, we build a single global DAG $G = (V, A)$ by introducing two dummy nodes: a unique start node α and a unique end node ω. The node α is connected to every source node of every job graph G_j, and every sink node is connected to ω. For each resource $r \in R$, we add an arc (α, ω)

that represents the (currently empty) sequence of operations on that resource. We assume a weight function $w : A \to \mathbb{R}_{\geq 0}$ assigning non-negative lags to all arcs, and that these lags satisfy the triangle inequality. With this modeling, we can represent shop scheduling problems with non-linear routes, multi-mode operations, sequence-dependent setup times, and, more generally, any problem with a regular objective whose solution can be expressed on a DAG with non-negative weights.

3 Overview of the insertion heuristic

The construction process in Algorithm 1 can be described as follows. Each iteration considers the full set of still unscheduled operations, i.e., $\mathcal{C} = \mathcal{E}$. This differs from most construction heuristics, which restrict attention to the currently *available* operations (those whose predecessors are already scheduled),i.e., $\mathcal{C} \subset \mathcal{E}$. The candidate operation is selected using a priority function π. In the computational results, the operation with the largest processing time is selected. For every candidate operation e and for every feasible mode $M \in \mathcal{M}_e$, we compute the set of feasible insertion positions induced by that mode and select the position that yields the best value of the objective. Once an insertion has been applied, the schedule data (heads and tails) must be updated. Recomputing longest paths from scratch at every iteration would make the heuristic too slow on large instances. Instead, we rely on the incremental update algorithm proposed in Katriel et al. (2005).

Algorithm 1 General Template of a Greedy Construction Heuristic

```
1: procedure buildSolution(ℰ, π)
2:     Initialize empty or partial schedule S
3:     while ℰ ≠ ∅ do
4:         Select candidate elements 𝒞 ⊆ ℰ eligible for scheduling
5:         e ← π(𝒞, S), ℰ ← ℰ \ {e}
6:         Determine set of feasible and admissible insertion positions ℐ_e
7:         Select an insertion position I_e ∈ ℐ_e for e
8:         S ← S ⊕ (e, I_e)
9:     return S
```

4 Efficient enumeration of insertion positions

For the heuristic to be effective and efficient, it must evaluate many candidate insertion positions quickly while avoiding the enumeration of dominated ones. This challenge is problem-dependent: in flow shop problems feasibility is essentially guaranteed, whereas in job shop problems every insertion may create a cycle in the conjunctive graph, so feasibility tests become the bottleneck. Early approaches such as Werner and Winkler (1995) test feasibility by actually inserting operations and checking for a cycle, which is computationally expensive when many positions must be checked. More efficient schemes are often used in the context of improvement heuristics. To solve the flexible job shop scheduling problem (FJSP) with makespan minimization, Mastrolilli and Gambardella (2000) propose an insertion scheme that exploits monotonicity of reachability along a machine sequence and combines constant-time sufficient conditions with binary search to determine only a small set of feasible and dominant positions. Kis (2003) later exploits the results of Mastrolilli and Gambardella (2000) to improve the simultaneous insertion approach of Brucker and Neyer (1998) on several machines. A similar approach is designed in Artigues and Roubellat (2002) to efficiently insert an operation in multiple sequences while considering maximum lateness. However, this last work shows that the introduction of sequence-dependent setup times invalidates the nice pruning properties, and it was proposed to use computationally expensive traversal algorithms to ensure feasibility of the insertion positions. It is also possible to build examples showing that the approaches of Mastrolilli and Gambardella (2000) and Kis (2003) can miss optimal positions for min-sum objectives such as weighted number of tardy jobs or total weighted tardiness.

To overcome these limitations we adopt the parameterized reachability procedure of Tamssaouet and Dauzère-Pérès (2023). The procedure asserts the absence (or presence) of a path between two nodes by running a breadth-first search with a cutoff depth; a path is confirmed or refuted during the traversal, otherwise sufficient conditions similar to those of the works above are evaluated using the deepest layer of the BFS tree. Because this check is parameterized, we can tune the cutoff to the difficulty of the instance: small cutoffs are sufficient for classical variants, while larger cutoffs ensure feasibility in the presence of constraints such as sequence-dependent setup times and dominance with regard to min-sum objectives. Coupled with efficient enumeration over machine sequences, this yields a unified mechanism that identifies feasible and dominant insertion positions for any regular objective, with a controllable trade-off between speed and accuracy.

4.1 Preliminary comparison with reference methods

This subsection reports a preliminary comparison between the proposed insertion-based construction heuristic and representative methods from the literature on four classes of scheduling problems: (i) the flexible job shop with non-linear routes (Birgin et al., 2015), (ii) the flexible job shop with multiple resources per operation (Dauzère-Pérès et al., 1998), (iii) the classical job shop problem (Mati et al., 2011), and (iv) the job shop with sequence-dependent setup times (Artigues et al., 2005). All problems are considered under a makespan objective, except for Mati et al. (2011) where the weighted number of tardy jobs is optimized. For each problem class we report the average upper-bound gap (UB gap, in %) with respect to the best upper bounds. Our goal is not to outperform every specialized method, but to assess how far a single, relatively problem-independent insertion heuristic stands from the *basic* and *advanced* procedures proposed in those papers. Table 1 summarizes the results. For every paper we identify two baselines:

- **Birgin et al.**: Birgin et al. (2015) propose a simple list algorithm (basic baseline) and a beam search (advanced baseline). The proposed heuristic clearly improves on the list algorithm. Its UB gap remains slightly above the beam search, but the running time is in the order of milliseconds, whereas the beam search reported in Birgin et al. (2015) requires more than 10^3 seconds on average.
- **Dauzère-Pérès et al.**: Dauzère-Pérès et al. (1998) also combine a list-based construction (basic) with a tabu search algorithm (advanced). The insertion heuristic outperforms the list construction while keeping the average CPU time below 0.5 seconds.

Table 1. Average UB gap (%) compared to reference methods.

Method	Birgin et al.	Dauzère-Pérès et al.	Mati et al.	Artigues et al.
Basic	20.3	129.4	185.9	8.2
Advanced	1.0	4.8	0.0	3.2
Insertion Heuristic	4.0	31.2	160.2	12.8 (6.3)

- **Mati et al.**: Mati et al. (2011) propose an insertion procedure (basic) and a tabu thresholding inspired method (advanced). Here again, the proposed heuristic is closer in spirit to the basic method and should be compared primarily with that baseline.
- **Artigues et al.**: Artigues et al. (2005) study a job shop with sequence-dependent setup times and combine six schedule generation schemes with nine priority rules. They run these combinations either once (single pass) or 1000 passes. The *basic* figure for Artigues et al. corresponds to the best results of all single-pass combinations, while the *advanced* figure corresponds to the best result over all SGS/priority-rule combinations when 1000 passes are allowed. The insertion heuristic, when executed once, yields an average UB gap of 12.8%, which is worse than their best single-pass configuration. When we run the heuristic 1000 times and keep the best solution, the average gap decreases to 6.3%, showing that the proposed construction method also benefits from multi-start.

Overall, these results suggest that a single insertion-based heuristic can outperform the basic/list-type constructions reported in three different studies, and can approach the quality of more elaborate methods when the latter rely on multiple passes or problem-specific improvements.

5 Conclusions

The proposed heuristic shows that a single, insertion-based constructor can remain both generic and efficient for shop scheduling problems that admit a DAG representation with non-negative arc weights and a regular objective. By combining a reachability-driven enumeration of feasible positions with an incremental update of heads and tails, it builds schedules that systematically outperform basic/list constructions from the literature and come reasonably close to more elaborate methods. This makes it a practical starting point for improvement heuristics and a small but concrete step toward more "generalist" scheduling heuristics that can be transferred across problem variants.

Bibliography

Artigues, C., Lopez, P., and Ayache, P.-D. (2005). Schedule generation schemes for the job-shop problem with sequence-dependent setup times: dominance properties and computational analysis. *Annals of Operations Research*, 138(1):21–52.

Artigues, C. and Roubellat, O. (2002). An efficient algorithm for operation insertion in a multi-resource job-shop schedule with sequence-dependent setup times. *Production Planning & Control*, 13(2):175–186.

Birgin, E. G., Ferreira, J. E., and Ronconi, D. P. (2015). List scheduling and beam search methods for the flexible job shop scheduling problem with sequencing flexibility. *European Journal of Operational Research*, 247(2):421–440.

Bräsel, H., Tautenhahn, T., and Werner, F. (1993). Constructive heuristic algorithms for the open shop problem. *Computing*, 51:95–110.

Brucker, P. and Neyer, J. (1998). Tabu-search for the multi-mode job-shop problem. *Operations-Research-Spektrum*, 20(1):21–28.

Dauzère-Pérès, S., Ding, J., Shen, L., and Tamssaouet, K. (2024). The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 314(2):409–432.

Dauzère-Pérès, S., Roux, W., and Lasserre, J. (1998). Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, 107(2):289–305.

Katriel, I., Michel, L., and Van Hentenryck, P. (2005). Maintaining longest paths incrementally. *Constraints*, 10(2):159–183.

Kis, T. (2003). Job-shop scheduling with processing alternatives. *European Journal of Operational Research*, 151(2):307–332.

Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

Mati, Y., Dauzère-Pérès, S., and Lahlou, C. (2011). A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research*, 212(1):33–42.

Nawaz, M., Enscore Jr, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

Pinedo, M. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing AG, Basel.

Tamssaouet, K. and Dauzère-Pérès, S. (2023). A general efficient neighborhood structure framework for the job-shop and flexible job-shop scheduling problems. *European Journal of Operational Research*, 311(2):455–471.

Werner, F. and Winkler, A. (1995). Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, 58(2):191–211.

Exact method for Solving Integrated Optimization Problems through a Bi-Level approach

Mallory Taffonneau[1], Yannick Kergosien[1], Marina Vinot[1], Ameur Soukhal[1]

Tours University, Laboratoire d'Informatique Fondamentale et Appliquée de Tours, équipe ROOT, 64 avenue Jean Portalis, 37200 Tours, France
{mallory.taffonneau, yannick.kergosien, marina.vinot, ameur.soukhal}@univ-tours.fr

Keywords: Scheduling, Vehicle Routing, Bi-Level Optimization, Exact Methods.

1 Introduction

Integrated scheduling and vehicle routing problems have received growing and sustained attention in recent years dues to both their scientific relevance and their practical applications (Berghman 2023). Their aim is to jointly coordinate production planning and distribution organization in order to minimize costs while meeting delivery deadlines. Although many studies focus on single-criterion or multi-criteria approaches, these studies generally assume that production and transportation decisions are made in a centralized manner or by stakeholders operating at the same hierarchical level. Such an assumption can be restrictive in real-world supply chain contexts, where scheduling and transportation activities are often handled by distinct entities with different decision-making levels.

In many real-world settings, the manufacturing company handles production planning, while distribution is outsourced to a transport provider. Upstream decisions such as scheduling sequences and product release times shape the operating conditions under which the carrier must construct its delivery routes.

This hierarchical interaction, where internal production optimization is combined with minimization of transportation costs, cannot be accurately captured by traditional integrated models, which generally assume a single compromise between objectives. In practice, each decision maker pursues their own objective function and responds to the decisions of the other. In this context, a bi-level modeling approach becomes particularly relevant to represent the resulting decision hierarchy. In a bi-level framework (Dempe 2002), a *leader* first selects a strategy optimizing its objective while constraining the subsequent decisions of a *follower*.

2 Problem description

In this study, we consider a production unit responsible for fulfilling a set $\mathcal{O}$ of customer orders. Once the orders are completed, their delivery is outsourced to an external carrier. In this bi-level setting, the production unit acts as the *leader* and seeks to minimize delivery delays, while the carrier assumes the role of the *follower* and aims to minimize their transportation costs. Each order is characterized an earliest possible production start date r_i (for instance, due to raw material availability), a processing time p_i, and a due date d_i set by the customer. The workshop consists of a single production line modeled as a single machine. Scheduling is carried out without preemption. Moreover, the production line operates continuously for efficiency reasons: no idle times may be deliberately inserted during the production horizon.

The *follower* provides a vehicle with limited capacity Q, that can perform multiple delivery tours, starting from a date agreed upon with the *leader*. Storage space at the loading dock is limited, as is commonly the case in warehouses. As a result, orders are

loaded onto the vehicle as soon as they are completed. When the vehicle is out on a tour, completed orders are temporarily stored at the dock and loaded upon its vehicle's return. After each tour, the vehicle returns to the workshop and immediately departs again with the newly available orders. With every couple (i, j) of customers is associated a distance $t_{i,j}$. For every tour, the carrier determines the delivery sequence in order to minimize the total distance traveled.

The schedule set by the leader largely determines the composition of each vehicle load, as the follower still retains some flexibility by adjusting the delivery tours in order to control the return time to the depot and, consequently, part of the composition of the subsequent tour. Once the follower has optimized the routes, the leader evaluates the total cost of delivery delays induced by the interaction between the two decision levels.

3 Bi-Level MILP Model

To formalize the problem, we propose a bi-level mixed-integer linear programming (MILP) model. The leader aims to minimize $\sum_{i\in\mathcal{O}} T_i$ where T_i is the tardiness of the order i to its delivery location. The leader's decisions are modeled using binary variables x_{ij}, equal to 1 if order i precedes order j on the machine, and 0 otherwise. Unlike most bi-level formulations in the literature, our model features a non-convex lower-level problem corresponding to the follower's decision process, which makes its resolution particularly challenging. The follower's objective is to minimize $\sum_{i\in\mathcal{O}}\sum_{j\in\mathcal{O}}\sum_{h\in\mathcal{T}} t_{ij}y^h_{ij}$, where y^h_{ij} is the follower decision variable, equal to 1 if vehicle delivers customer i then j during trip h, and 0 otherwise.

Most commercial LP solvers, such as CPLEX or Gurobi, are not suited for direct bi-level model resolution. Initial tests confirmed their performance is inadequate for our problem. Dedicated tools exist (e.g., (Fischetti 2017)), but its application to our model quickly revealed significant limitations. These observations motivate developing a tailored exact method capable of handling larger instances while guaranteeing optimality.

4 Branching Search Procedure

To overcome the limitations of the MILP approach, we propose an exact solution method tailored to the specific characteristics of our bi-level problem. The main idea is to solve the scheduling problem (leader problem) using a Branch-and-Bound (B&B) procedure that explores the space of all possible order permutations. At each relevant leaf node of the search tree, the corresponding routing problem (follower problem) is then solved exactly using a "Label Correction Procedure". We now present the main components of the proposed method.

Branching Strategy and Node Selection: Let $\mathcal{O}$ denote the finite set of customer orders. A partial schedule is represented by a sequence: $\sigma = (J_{[1]}, \ldots, J_{[i]})$, $i \in \{1, \ldots, |\mathcal{O}|\}$, where $J_{[i]}$ denotes the i^{th} order. The root node is associated to $\sigma = \emptyset$.

At each branching step, an unscheduled order $J_i \in \mathcal{O} \setminus \sigma$ is appended to the end of the sequence, generating a child node $\sigma' = (\sigma, J_i)$.

Nodes are explored in increasing order of their lower bound $LB(\sigma)$. When multiple nodes share the same bound, they are explored according to a depth-first strategy. A LB is computed only if the completion time induced by σ exceeds the departure time of the first vehicle tour.

Lower Bound: The lower bound $LB(\sigma)$ provides a tight underestimation of the minimal total tardiness achievable from a given node. It is calculated as the sum of two parts:

1. Tardiness of Scheduled Orders $T(\sigma)$: To estimate the tardiness of the current sequence σ, we determine how orders are partitioned into delivery trips. Since the optimal routing is not yet known, we apply a recursive procedure that evaluates every feasible trip partition $s \in \mathcal{S}$. For each partition s, the departure time of trip h, denoted $S^{h,s}$, is defined as the earliest time at which all orders assigned to trip h have been completed and the vehicle has returned from the previous trip. The return time from trip $h-1$ can be estimated by $S^{h-1,s} + 2 \times max_{i \in \mathcal{H}^s_{h-1}}(t_{0,i})$ where $\mathcal{H}^s_{h-1}$ denotes the set of orders assigned to trip $h-1$ in partition s. The earliest delivery time of customer i assigned to trip h is $D^{h,s}_i = S^{h,s} + t_{0,i}$. Tardiness T^s_i of each order for the partition s is computed as $T^s_i = max(0; D^h_i - d_i)$ where d_i is the due date of order i. Finally, the value T_σ is defined as the minimum total tardiness over all feasible trip partitions.
2. Tardiness of Unscheduled Orders: for the remaining orders, we adopt a direct-delivery assumption to estimate their contribution to tardiness. We assume each order $i \in U(\sigma)$ is delivered individually as soon as it is completed, either after is release date (r_i) or immediately after the completion time of the last order in σ (C^σ_{max}). The estimated tardiness for each remaining order is $T_i = max(C^\sigma_{max}, r_i) + p_i + t_{0,i} - d_i$. Hence the overall lower bound is the sum of these two parts: $LB(\sigma) = T_\sigma + \sum_{i \in U(\sigma)} T_i$.

Upper Bound Initialization: The upper bound UB is initialized using two complementary heuristics: i. *Earliest Due Date (EDD)* where orders are sorted by increasing order of their due dates, providing a deterministic initial sequence; ii. *Random sampling* where ten random permutations of the orders are generated. For each permutation, the follower's routing problem is solved optimally through the label correction procedure described below, allowing to obtain the final bi-level solution. The best evaluated sequence defines the initial UB.

"*Label Correction Procedure*" to solve follower problem: At each leaf node, when all orders in σ have been scheduled, the routing problem is solved optimally using a label correction procedure on a graph in which nodes correspond to the orders in the sequence σ. An arc (i, j) is added if order J_i precedes order J_j in σ and if the total load of all orders between J_i and J_j in the sequence does not exceed capacity Q of vehicle. A label associated to a node i is characterized by the arrival time at the production site and the accumulated routing cost. Each label is propagated along feasible arcs, taking into account arrival times and the release constraints imposed by the production sequence. For each propagation, multiple new labels are generated, corresponding to the possible delivery tours. Dominance rules are applied only between labels with identical arrival times or complete sets of tours, and only based on the routing cost. Symmetries, notably palindromic tours, are eliminated, reducing roughly half of the solution space. The algorithm continues until no further label extension are possible. The best non-dominated terminal label at the node of the last order provides the optimal routing cost for the complete schedule σ. Thus every leaf node is evaluated exactly, guaranteeing bi-level optimality of the solution.

5 Experimental results

Experiments were conducted on 20 instances for each problem size (5, 10 and 15 orders). All the approaches ran on a computer 8-core with processor AMD Ryzen 7 7800X3D, 4.20 GHz and 32 GB of RAM and a time limit of 3600 seconds (1 hour). Instances are generated as follows. Customers are first placed randomly according to a uniform distribution within

a 16×16 square, with the production unit located at the center. Travel times d_{ij} between all locations are then computed and rounded down to the nearest integer. Processing times p_i are drawn uniformly at random from $[1, 10]$. Release dates r_i are sampled uniformly between 0 and $\sum_{i \in \mathcal{O}} p_i$, while due dates d_i are drawn uniformly from $[r_i + p_i + 16, 2(r_i + p_i + 16)]$, where 16 corresponds to the size of the square. Order sizes q_i are uniformly sampled between 10 and 20, and the vehicle capacity is set to $Q = \frac{3 \times \sum_{i \in \mathcal{O}} q_i}{n}$.

Table 1. Results of both exact methods on a set of instances of size 5, 10 and 15

	MILP				**Branching Search**			
n	**#opt**	**time(s)**	**gap(%)**	**#feas**	**#opt**	**time(s)**	**gap(%)**	**#feas**
5	20	1.232	0	20	20	0.002	0	20
10	1	3436	95	18	20	62	0	20
15	0	3600	100	0	2	3296	69	20

In Table 1, the columns report: $\#opt$ the number of instances for which the optimal solution was found; gap the percentage deviation relative to the best solution found and LB; and $\#feas$ the number of instances for which a feasible solution was obtained, without guaranteeing optimality.

As shown in Table 1, the branching search outperforms the MILP solver on small instances, especially those with 10 orders. In these cases, all optimal solutions are found within a few minutes, whereas the MILP solver requires up to one hour and finds only a single optimal solution. For size $n = 15$, the branching search method successfully identified feasible solutions for all 20 instances and two optima, whereas the MILP solver failed to find any feasible results within the time limit.

6 Conclusion and perspectives

We addressed an integrated production within a bi-level modeling framework. Branch search algorithm was proposed, and computational experiments on small instances yielded encouraging results, highlighting the potential for obtaining bi-level optimal solutions.

For future work, a major avenue for improvement lies in strengthening the LB. Promising directions include deriving more accurate bounds based on combinatorial structures, such as those proposed in (Hougardy 2014), and integrating dominance conditions between nodes in the search tree to prune infeasible or suboptimal sequences earlier.

References

Berghman L., Kergosien Y., Billaut J-C., 2023, "A review on integrated scheduling and outbound vehicle routing problems" *European Journal of Operational Research*, 311(1), 1-23.

Chevroton H., Kergosien Y., Berghman L., Billaut J. C., 2021, "Solving an integrated scheduling and routing problem with inventory, routing and penalty costs" *European Journal of Operational Research*, 294(2), 571-589.

Dempe S., 2002, "Foundations of bilevel programming", Boston, MA: Springer US.

Fischetti M., Ljubić I., Monachi M., Sinnl M., 2017, "Algorithm for mixed-integer bilevel linear programs", *Operations Research*, vol. 65, no 6, pp. 1615-1637.

Hougardy, S., 2014, "On the integrality ratio of the subtour LP for Euclidean TSP" *Operations Research Letters*, 42(8), 495-499.

Kleinhert T., Labbé M., Ljubić I., Schmidt M., 2021, "A Survey on Mixed-Integer Programming Techniques in Bilevel Optimization." *EURO Journal on Computational Optimization*, vol. 9, 100007

O - Heuristics and metaheuristics #2

A graph neural network based decomposition approach for the workload balancing problem

Alice Poboni[1], Giulio Modesti[1], Andrea Gasparin[1], Valentina Blasone[1], Alex Dagri[1] and Lorenzo Castelli[1]

University of Trieste, Italy
{alice.poboni, giulio.modesti}@studenti.units.it
andrea.gasparin@dia.units.it, {valentina.blasone, alex.dagri}@phd.units.it,
castelli@units.it

Keywords: workload balancing, jop shop scheduling, graph neural network, heuristic, optimisation

1 Introduction

The classical workload balancing (WB) problem in scheduling (Shtmoyashiro, Tsoda and Awane 1984, Cigolini, Portioli-Staudacher 2002) focuses on optimally allocating tasks across resources to minimise imbalances, thus enhancing system efficiency by preventing both overload and underuse. This challenge is particularly critical for large enterprises operating distributed production networks. In such contexts, the optimisation goal extends beyond the traditional minimisation of total production time (makespan) or total job delay. A key additional objective is to balance the workload across the various production facilities. This is essential for managing critical resources, such as specialised machinery or skilled personnel, which often require a specific, sustained workload level to function at peak operational and economic efficiency. This introduces a multi-criteria optimisation problem that integrates scheduling with strategic resource utilisation across a network. The WB can be formalised as a Mixed Integer Linear Programming problem, but in many real-world scenarios, the large number of tasks, facilities, and resources make the problem intractable for traditional solvers; hence, there is a need to develop efficient heuristics. One of the main approaches is represented by constructive heuristics (Perez-Gonzalez, Fernandez-Viagas, Zamora García and Framiñán 2019) based on dispatching rules which consist of priority rules to select the best next assignment job-resource to do at each time step. Although these algorithms provide feasible solutions without requiring great computational effort, they generally do not provide high quality solutions.

To overcome these challenges, this work proposes a novel divide-and-conquer solution approach that combines exact optimisation methods with machine learning techniques. This hybrid framework is designed to efficiently tackle large-scale instances of the problem where traditional monolithic solvers fail. Rather than addressing the entire problem directly, our method employs a strategic iterative refinement process. It begins by constructing an initial feasible solution using a fast greedy heuristic. The core of the algorithm then iteratively selects and solves optimally sized sub-problems, each considering a small subset of facilities. This procedure repeats until a predefined stopping criterion is met.

However, for rapid convergence to high-quality solutions, the sub-problem selection (performed at each iteration) plays a crucial role. To this end, we leverage a Graph Attention Network (GAT), trained to predict which subset of facilities, once locally re-optimised, is expected to yield the greatest global improvement in the objective function. This learned heuristic guides the decomposition, ensuring that computational effort is focused on the most promising areas of the solution space.

Extensive computational experiments show how our approach overcomes the scalability limitations of exact solvers like Gurobi, consistently delivering high-quality solutions with stable performance as the instance size increases.

2 Problem formulation

Given a set of facilities F, a set of resources K_i for each facility $i \in F$, a set of critical resources $\mathcal{K}_i \subseteq K_i$ for each facility $i \in F$, and a set of jobs J, the Multi-Facility Load Balancing Problem consists of finding a feasible assignment of jobs to facilities and time periods that respects resource capacities while minimising a weighted sum of total tardiness and imbalance across critical resources. In this framework, we assume that each job can be performed by all facilities and that each facility can perform multiple jobs in the same time period. We define the decision variables as follows. Let $x_{i,j,t} \in \{0,1\}$ equal to 1 if job $j \in J$ is assigned to facility $i \in F$ and processed in time period $t \in T$, and 0 otherwise; $d_j \geq 0$ denote the tardiness of job $j \in J$; and $y_{k,t} \geq 0$ represents the total load imposed on resource $k \in K_i$ at facility $i \in F$ during time period $t \in T$.

$$\min \quad \alpha \sum_{j \in J} d_j + \beta \Big(\sum_{\substack{i \in F \\ k \in \mathcal{K}_i \\ t \in T_1}} |w_{k,t} - y_{k,t}| + \sum_{\substack{i \in F \\ k \in \mathcal{K}_i \\ t \in T_2}} \max(0, y_{k,t} - w_{k,t}) \Big) \tag{1}$$

$$\text{s.t.} \quad \sum_{i,t} x_{i,j,t} = 1 \quad \forall j \in J \tag{C1}$$

$$\sum_{j} x_{i,j,t} \cdot p_{j,k} = y_{k,t} \quad \forall i \in F, \forall k \in K_i, \forall t \in T \tag{C2}$$

$$y_{k,t} \leq c_{k,t} \quad \forall i \in F, \forall k \in K_i, \forall t \in T \tag{C3}$$

$$\sum_{t \in T} x_{i,j,t} \cdot t \leq T_j + d_j \quad \forall i \in F, \forall j \in J. \tag{C4}$$

Constraint C1 ensures the assignment of all jobs. C2 quantifies the load of each resource $k \in K_i$ of the facility $i \in F$ at each time period (where $p_{j,k}$ is the load required by k to process j), and C3 ensures that its value must respect the resource's capacity $c_{k,t}$, which might variate depending on the time period. Finally, C4 quantifies the delay (if any) of each job j with respect to its due date T_j. The first term of the objective function (1) represents the overall delay. The second, instead, the deviation from the optimal workload of all critical resources, which is further divided into two components. The first component accounts for deviation during the regular period T_1, while the second component applies to the additional buffer period T_2, during which underload is not penalised. While not linear in its current form, the objective function can be linearised through the introduction of auxiliary variables and a small set of additional constraints.

3 Our Approach

We propose a hybrid divide-and-conquer framework that iteratively decomposes and refines the multi-facility load balancing problem. The method alternates between exact optimisation of small sub-problems and efficient selection of which sub-problems to solve next, guided by a machine learning model. The main steps of our approach are summarised as follows:

Initial Solution. A constructive heuristic (EDD and assignment to the available facility with the shortest processing time) provides a feasible schedule, i.e., the starting point.

Decomposition. The full problem ($|F|$ facilities over $|T|$ periods) is split spatially and temporally. The horizon is divided into Θ contiguous segments, creating Θ copies of each facility. Each iteration selects a small subset of (facility, time-segment) pairs together with their assigned jobs, forming a tractable sub-problem.

Iterative Optimization. At each iteration, we extract the current assignments for the selected subset and formulate the corresponding sub-problem as a MILP. This restricted problem is solved to optimality using a commercial solver. Because the sub-problem is small, the solver finds the provably optimal reassignment of jobs within this subset very quickly-often in seconds. Reintegrating this improved partial solution guarantees that the global objective function value does not increase. The process repeats, gradually improving the overall schedule through a sequence of localised optimisations.

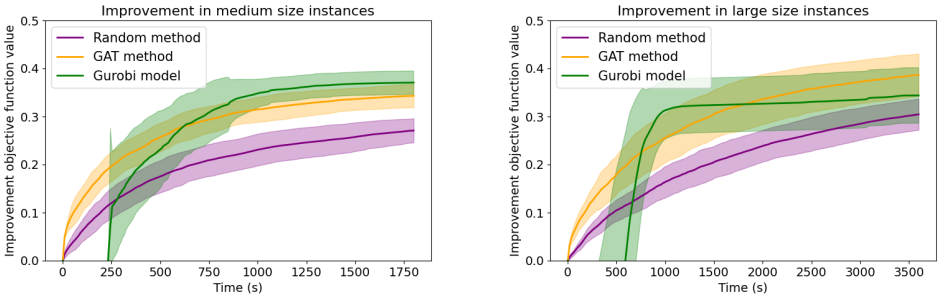
3.1 Selection via GAT

The choice of which (facility, time period) pairs to optimise critically affects convergence speed. To make this selection efficiently, we model the current state as a fully connected graph. Each node represents a unique (facility, time period) pair, with node features encoding its current load, capacity, processing-time statistics, penalties related to its job assignments. Edges encode relationships between the pairs, including job punctuality, similarity in processing times, coordination on critical resources, and correlations in their temporal processing patterns. A pre-trained GAT processes this graph and assigns a scalar score to each edge, which represents the normalised improvements in the global objective if the two connected nodes were jointly optimised. The edge with the highest score (highest predicted improvement) is selected for the next optimisation step. After each step, node features are updated to reflect the new schedule. The GAT is trained offline in a supervised fashion on data generated by an exhaustive greedy oracle. For each problem instance, the oracle (the exact solver) computes the true objective decrease for every possible edge; these values are normalised as target labels.

4 Results

We compare our approach against two benchmarks: the exact formulation and a random variant of our algorithm (Section 3) that selects sub-problems uniformly.
We test 20 medium instances (12 facilities, 20 periods, 30 min time limit) and 20 large instances (16 facilities, 28 periods, 60 min time limit), with $\Theta = 4$ sub-periods. Resources per facility follow a $\text{Bin}(4, 0.5)$ distribution. The $1300 * |F|$ jobs have due dates uniformly distributed over 52 periods; job loads are uniform between 105-165 min for critical resources and 90-135 min for others. The code has been implemented in Python, using the PyTorch Geometric library for the GAT and Gurobi for the exact solver. For mesuring the solution quality we considered the improvement (in percentage) of the solution with respect to the initial solution obtained with the greedy heuristic. The GAT has been trained on 900 problems solved to optimality. All experiments have been performed on a Intel(R) Core(TM) i7-14700K 3.40 GHz CPU computer (Windows 11, Python 3.13.9, Gurobi Optimizer version 13.0.0 build v13.0.0rc1) with 32GB RAM.

As shown in figure 1a, in the medium size instances, after half an hour of computation our method almost matches the performance of Gurobi, providing an average gap of 4.4% to the Gurobi best solution (3.0% standard deviation), and outperforms the random method, which reaches an average gap of 15.8% (3.8% standard deviation).For the large size instances our method outperforms also Gurobi solver (figure 1b), improving its solutions by the 6.2% on average (8.0% standard deviation). Instead, the random method remains on average 6.4% worse than Gurobi (7.8% standard deviation).



(a) (b)

Fig. 1: Results on instances of medium and large size

Figures 1a and 1b show that our approach finds high-quality solutions faster than Gurobi, which requires significant time to initialise and find a first feasible solution, and in particular in the large case, our method shows more consistent improvement. Furthermore, in these experiments, we have focused on instances that remain within Gurobi's solvable range. However, for larger problems (more factories and longer planning horizons) the memory requirements for Gurobi could become prohibitive. In such cases, our decomposition-based strategy remains viable, enabling the solution of problems beyond the reach of conventional solvers.

5 Conclusions

In this work we developed a machine learning-based heuristic using a divide-and-conquer approach to tackle the MFLBP, effectively addressing large real-world instances where exact solvers fail. Our method consistently outperforms a random baseline, particularly as instance size increases, demonstrating the efficacy of our refined graph neural network scoring mechanism for sub-problem selection. Comparative analysis against the Gurobi solver across medium, and large instances revealed a key strength of our heuristic: scalability. While Gurobi performance degraded significantly with size, suffering from high runtime memory and large optimality gaps, our heuristic maintained stable performance, finding high-quality solutions faster and more reliably at scale.

In conclusion, the proposed heuristic provides a robust, scalable, and effective approach for large-scale MFLBP instances.

References

Shtmoyashiro, Tsoda and Awane 1984, "Input scheduling and load balance control for a job shop", *International Journal of Production Research*, 22(4), pp. 597-605.

Cigolini, Portioli-Staudacher 2002, "An experimental investigation on workload limiting methods within ORR policies in a job shop environment", *Production Planning & Control*, 13(7), pp. 602-613.

Perez Gonzalez, P., Fernandez Viagas, V., Zamora García, M. and Framiñán, J.M. 2019 "Constructive heuristics for the unrelated parallel machines scheduling problem with machine eligibility and setup times", *Computers & Industrial Engineering*, 131, pp.131-145.

Integrated Scheduling, Maintenance, and Sampling Decisions for Risk Reduction in Semiconductor Manufacturing

Julien Autuori[1], Stéphane Dauzère-Pérès[1] and Claude Yugma[1]

Mines Saint-Etienne, Univ. Clermont Auvergne, CNRS, UMR 6158 LIMOS
Gardanne, France
{julien.autuori,dauzere-peres,yugma}@emse.fr

Keywords: Scheduling, Semiconductor Manufacturing, Risk Minimization, Genetic Algorithm.

1 Introduction and Related Work

Semiconductor manufacturing is a long and highly complex process in which lots (jobs) of wafers are routed through multiple workshops comprising numerous processing steps (or operations) (Mönch Lars *et. al.* 2011) on production machines. Thus, each job requires a given number of consecutive operations, called a route. Jobs are also regularly controlled on measurement tools. Both production machines and measurement tools have limited capacity. Hence, measuring every job is infeasible, and manufacturers rely on selective sampling. A positive measure (i.e., the products in the lot are not defective) ensures that all products processed on the same machine since the positive measure are not defective. However, because the operations of a job occur at different times and on different machines, only jobs processed earlier on the same machine can be validated, creating a strong coupling between scheduling, maintenance, and sampling decisions.

This study aims to minimize the risk of product defects by jointly coordinating these three types of decisions. Although each type has been extensively studied, fully integrated approaches remain scarce. Surveys on sampling strategies (Nduhura-Munga Justin *et. al.* 2013) report methods ranging from static sampling with fixed sampling rates to dynamic sampling that explicitly considers the capacity of measurement tools. Health indices offer a compact representation of machine degradation and support maintenance and quality decisions (Kao Yu-Ting *et. al.* 2018), often derived through data-driven techniques such as spectral clustering or autoencoder-based modeling. Metaheuristics have also been developed for scheduling with maintenance constraints, e.g., in (Tamssaouet Karim *et. al.* 2018) for the job-shop scheduling problem (JSP).

Departing from these individual lines of work, this study adopts a fully integrated perspective by jointly optimizing production scheduling, maintenance, and sampling decisions. We use a health-based indicator, the weighted WAR (Wafer-At-Risk) (Nduhura-Munga Justin *et. al.* 2013), which evaluates the risk of product defects in each job due to machine degradation while simultaneously guiding maintenance decisions. The resulting framework aims to minimize the overall risk and to detect potential issues on production machines as early as possible. The contribution is an integration of the JSP and risk default.

2 Problem Formalization and Resolution Methods

The workshop under study consists of several production machines and a single measurement tool. We assume that all measures are positive, i.e., no defective products are detected. Otherwise, another process starts to locate, stop and maintain the defective production machine. The measurement tool processes one job at a time.

We consider a Job-Shop Scheduling Problem (JSP). Two sets are considered: the set of production machines and the set of jobs. Each job follows an ordered sequence of operations, called a route, where each operation needs to be performed on a given machine. In this study, all jobs visit all machines, though not necessarily in the same order or with identical processing times.

2.1 Dynamic Behavior of the System

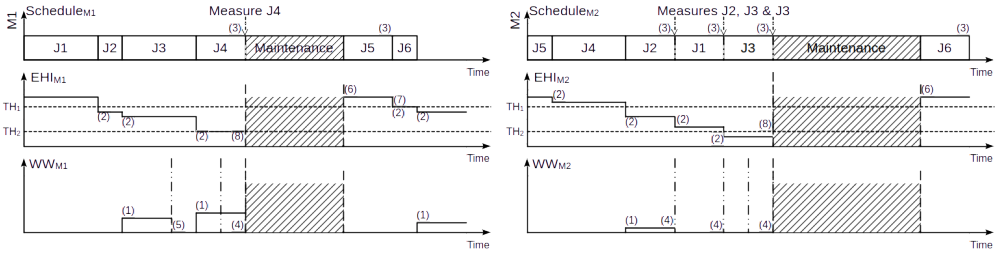


Fig. 1: Evolution of the variables

Figure 1 illustrates the evolution of key variables for two machines ($M1$ and $M1$) and six jobs. Each machine is associated with three dynamic components:

1. The *schedule* with the sequence of operations and maintenance actions,
2. The *Equipment Health Index* (EHI) that models the machine degradation, and
3. The *Weighted War* (WW), that captures the accumulated risk.

The weighted war (WW) of machine m increases at the completion time of an operation proportionally to the EHI at its start (1), while the EHI decreases while the job is being processed (2). A job can be measured once the last operation in its route is completed (3). A positive measure resets the WW of the machine performing the last operation (4) and partially reduces the WW on other machines previously visited by the job (5). The reduction corresponds to the risk level when the job left each machine.

Maintenance actions are performed to prevent the EHI of machines to reach zero, i.e. to avoid a machine failure. When maintenance is performed, the EHI returns to its maximum value (6). Measures and maintenance are governed by two thresholds: A measure is triggered when the EHI is smaller than TH_1 (7), whereas maintenance occurs only when it is smaller than TH_2 (8). These thresholds avoid unnecessary corrective maintenance operations and ensure stable system behavior.

The objective is to minimize the cumulative risk, expressed as:

$$\mathcal{F}_1 = \sum_{m \in M} \sum_{t \in H} w_{m,t},$$

where $w_{m,t}$ denotes the WW of machine m at period t, M is the set of machines, and H is the scheduling horizon.

2.2 Solution Approaches

Two complementary solution approaches are proposed: (1) An integer linear program that can only solve small instances, and (2) A Genetic Algorithm (GA) for solviong industrial-sized instances.

Figure 2 presents the chromosome structure. The first genome (1) encodes the job schedule following a classical JSP representation. A second genome (2) specifies the action

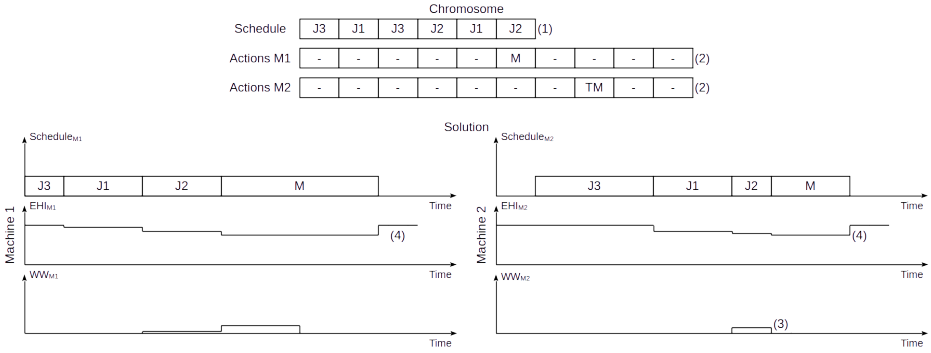


Fig. 2: Chromosome and corresponding decoded solution

assigned to each machine over time, choosing among four possibilities: (1) **-**, no action (normal production), (2) **T**, measurement of a job that has completed its last operation, (3) **M**, maintenance, and (4) **TM**, combined measurement and maintenance.

The decoding process is illustrated in Figure 2. For each machine, the first graph displays the sequence of operations and maintenance operations, the second graph shows the evolution of the EHI, and the third graph depicts the WW. During a measure (3), the WW decreases, while maintenance operations (4) restore the EHI to its maximum value.

3 First Numerical Results

Computational experiments were conducted on an HP EliteBook 840 G11 with an Intel® Core™ Ultra 5 125U processor and 32 GB of RAM running Ubuntu 24.05.2 LTS. All implementations were developed in C++ using the CPLEX API. Instances were randomly generated using a uniform distribution. The input parameters are the number of jobs, the number of machines, and the scheduling horizon. For each instance, the generator defines the routes of jobs, the maintenance durations, and the processing times. The route of each job is explicitly specified to ensure instances consistent with industrial conditions. Two groups of instances are considered. The first group includes *small* instances used to validate the mathematical model and verify that the GA gives optimal solutions. These include two machines with a scheduling horizon of 10, with ten instances of two jobs and ten instances of three jobs. The second group consists of *industrial-sized* instances with 10 machines, 50 jobs, and a scheduling horizon of 250. The processing times of operations range from 1 to 3 time units, and of maintenance operations from 2 to 6. The EHI of each machine decreases by 0.5 units per unit of processing time.

Table 1 reports the numerical results for the small instances. For each instance size, the optimal solution determined by the mathematical model is compared with the best solution determined by the GA. The mathematical model solves all instances to optimality within 15 minutes, and the GA reaches the same solutions in less than 1,000 iterations.

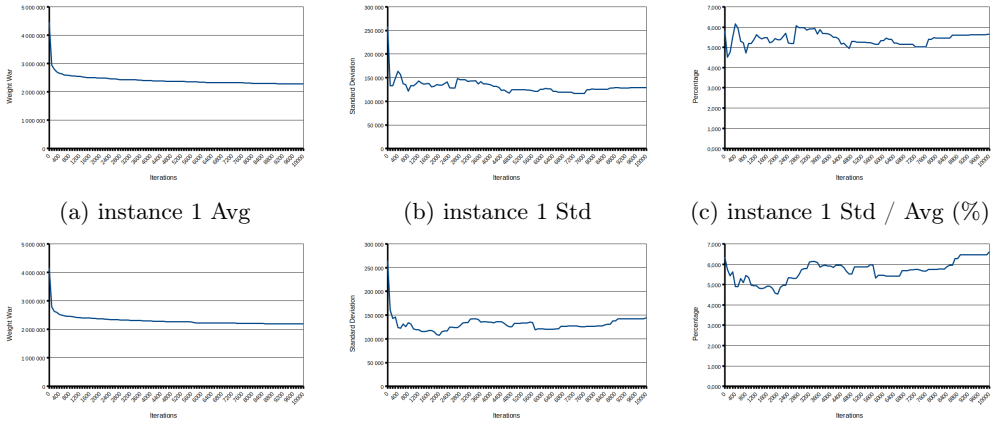
Table 1: Optimal results for small instances

Stop		2/2/10									2/3/10									
Model	15 min	450	450	700	650	700	500	700	650	600	550	950	650	500	800	1000	550	900	1000	550	1350
GA	1000 iter	450	450	700	650	700	500	700	650	600	550	950	650	500	800	1000	550	900	1000	550	1350

For the two industrial-sized instances, 10 independent runs were executed. Figure 3 summarizes the results, with the x-axis showing the number of iterations (up to 10,000, sampled every 100 iterations). Figures 3a–3d present the average WW over the 10 runs,

while Figures 3b–3e show the corresponding standard deviation. Figures 3c–3f report the relative standard deviation (standard deviation divided by the mean).

Most improvements occur within the first 1,000 iterations. Beyond this point, the objective stabilizes and variability across runs remains low. The relative standard deviation remains close to 6% for both instances, indicating good algorithmic robustness. Average computational times were 225–227 seconds for 100 iterations, 2052–2167 seconds for 1,000 iterations, and 21,597–22,778 seconds for 10,000 iterations.



(a) instance 1 Avg (b) instance 1 Std (c) instance 1 Std / Avg (%)

(d) instance 2 Avg (e) instance 2 Std (f) instance 2 Std / Avg (%)

Fig. 3: Numerical results with GA for industrial-sized instances

4 Conclusion

In this work, an integrated approach that jointly optimizes production scheduling, maintenance, and sampling decisions in semiconductor manufacturing is proposed. By incorporating a health-based risk indicator, the approach explicitly links equipment degradation to quality control actions. An integer linear programming model only solves small instances to optimality, while the proposed GA efficiently handles industrial-sized instances and provides stable results across runs. Overall, the study shows that coordinated decision making can effectively reduce risk and improve operational robustness.

References

Mönch Lars, Fowler John W., Dauzére-Pérès Stéphane, Mason Scott J. and Rose Oliver, 2011, "A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations", *Journal of Scheduling*, Vol. 14, pp 583-599.

Kao Yu-Ting, Dauzère-Pérès Stéphane, Blue Jakey and Chang Shi-Chung, 2018, "Impact of integrating equipment health in production scheduling for semiconductor fabrication", *Computers & Industrial Engineering*, Vol. 120, pp 450-459.

Nduhura-Munga Justin, Rodriguez-Verjan Gloria, Dauzère-Pérès Stephane, Yugma Claude, Vialletelle Philippe and Pinaton Jacques, 2013, "A literature review on sampling techniques in semiconductor manufacturing", *IEEE Transactions on Semiconductor Manufacturing*, Vol. 26, pp 188-195.

Tamssaouet Karim, Dauzère-Pérès Stéphane, and Yugma Claude, 2018, "Metaheuristics for the job-shop scheduling problem with machine availability constraints", *Computers & Industrial Engineering*, Vol. 125, pp. 1-8.

Towards a description of correlations between heuristic parameters for a scheduling problem

Chahboub R, Chemla J, Raveaux R, T'Kindt V

Laboratoire d'Informatique Fondamentale et Appliquée de Tours (LIFAT), France
{racha.chahboub, jean-paul.chemla, romain.raveaux,tkindt}@univ-tours.fr

Keywords: Scheduling, ant colony optimization, parameter optimization.

1 Introduction

In Operations Research (OR), algorithms generally rely on some parameters that need to be specified in advance. Their values are crucial since they have a big impact on their efficiency. In practice, these values are determined empirically and manually. Finding a "suitable" collection of parameters is made possible by this laborious process, which typically does not ensure the best choice of parameters. The process is complicated by the fact that the ideal set of parameters of an algorithm varies depending on the instance that is solved (Doerr & Doerr 2020).

Although automated tuning is widely used in Artificial Intelligence (AI) and machine learning (ML), its adoption in Operations Research (OR) is still at an early stage. A number of recent studies have shown a growing interest in parameter tuning and parameter control, whether dynamic or static (Doerr & Doerr 2020). Most automated configuration approaches focus on identifying a single high-performing configuration over a set of instances. In contrast, our work aims at deriving parameter values that depend explicitly on instance characteristics. We investigate how the different parameters are correlated and how they influence the efficiency of the algorithm. The focus is set on the solution of scheduling problems.

For our experiments, we consider a two-machine flowshop scheduling problem with n jobs. Each job $j \in \{1, \ldots, n\}$ has a processing time $p_{j,1}$ on machine 1 and $p_{j,2}$ on machine 2. Let C_j be the completion time of job j on the second machine. We focus on two criteria: the makespan C_{max}, defined as the maximum completion time of jobs on machine 2 and the total completion time $\sum C_j$, defined as the sum of completion times of jobs on machine 2.

The problem denoted by $F2||Lex(C_{max}, \sum C_j)$ consists in finding a schedule that minimises the sum of completion times under the condition that the makespan is optimal.

Problem $F2||C_{max}$ is polynomial, and solvable by Johnson's algorithm (Johnson 1954), while problem $F2||\sum C_j$ is strongly $\mathcal{NP}$-hard. As a result, the combined lexicographic problem $F2||Lex(C_{max}, \sum C_j)$ is also $\mathcal{NP}$-hard (T'kindt & Billaut 2006).

This problem has been the subject of multiple studies over the years, and both exact methods (T'kindt et al. 2003) and heuristics (Gupta et al. 2001, T'kindt et al. 2002, T'kindt et al. 2003) have been proposed. Among them, the SACO heuristic introduced in (T'kindt et al. 2002) is very efficient and based on Ant Colony Optimisation. In their publication, the authors suggest values for the parameters (evaporation coefficient of 0.9, number of ants $Nb_{ants} = 20$ and number of iterations $Nb_{iter} = 100$) based on "preliminary experiments".

Our contributions are the following: first, in Section 2, we propose a method to automatically determine values for the parameters that maximise the efficiency. Next, in Section 3, we investigate existing correlations between the parameters and the impact of the instances of the scheduling problem on these correlations.

2 Bayesian optimisation for ACO tuning

A popular approach for global black-box function optimisation is Bayesian optimisation, which is primarily used in ML. Bayesian optimisation is a technique that aims at computing the maximum of a mathematically unknown function f. The only requirement is to have an *oracle* capable of computing the value $f(x)$ for any input $x \in \mathbb{R}^N$. Assuming that the function can be modelled by a Gaussian process, this technique iteratively builds an approximation of f. At each iteration, an *acquisition function* is used to determine the next point x^t on which the *oracle* is called to compute $f(x^t)$. Then, $(x^t, f(x^t))$ is used to update the approximation of the function f. This process avoids exploiting the solution space in a brute-force way to derive x^*, the solution that maximises f.

Bayesian optimisation can be used to learn a good set of parameters for the SACO heuristic on the scheduling problem. The parameters are the evaporation coefficient EC, the number of ants Nb_{ants}, and the number of iterations Nb_{iter}.

A first question emerges: which learning model is suitable for predicting these parameters? Since some problem characteristics may influence their value, we formulate the preliminary hypothesis that the instance size n is the most significant factor. Moreover, we focus on configuring only Nb_{ants} and Nb_{iter}, as the evaporation coefficient is fixed at 0.9 following consistent evidence in the literature that this value performs well (T'kindt et al. 2002). Under these assumptions, we model the parameters as linear functions of n: $Nb_{ants} = \alpha_1 \times n + \alpha_2$ and $Nb_{iter} = \beta_1 \times n + \beta_2$.

Thus, using the above notations for Bayesian optimisation, we set $x = [\alpha_1, \alpha_2, \beta_1, \beta_2]$. The function to maximise is then defined as the average efficiency of the SACO heuristic for a given x, on a learning database B_{learn} of scheduling instances that are randomly generated. We also generate a separate validation database B_{val} to verify that the learned parameters generalise to unseen instances. More specifically, let us denote by $\sum_j C_j^A(I)$ (resp. $\sum_j C_j^{bA}(I,x)$) the value of the objective function computed by the SACO heuristic with the default values (resp. with parameter values x). On a given instance I of the scheduling problem, we have:

$$f(x) = \sum_{I \in B_{learn}} \frac{\sum_j C_j^A(I) - \sum_j C_j^{bA}(I,x)}{\sum_j C_j^A(I)} \tag{1}$$

Table 1. Performance comparison between default and learned SACO parameters

	Baseline				**Learned**			
n	$t_{avg}(s)$	$dev_{avg}(\%)$	Nb_{ants}	Nb_{iter}	$t_{avg}(s)$	$dev_{avg}(\%)$	Nb_{ants}	Nb_{iter}
50	0.44788	0.07038	20	100	1.04573	0.01278	63	122
100	3.012984	0.14222	20	100	20.50486	0.00273	113	170
150	8.66312	0.13803	20	100	126.33169	0.00027	163	219

The application of the Bayesian optimisation led to coefficients $\alpha_1 = 0.99683, \alpha_2 = 13.78554, \beta_1 = 0.96902$ and $\beta_2 = 74.08415$, after 50 iterations corresponding to three days of computation. Table 1 presents a computational comparison on B_{val} of the baseline SACO (i.e. with the default parameters) and the SACO heuristic with the learned parameters. The columns t_{avg} report the average running time, in seconds, computed over 50 instances for each problem size. The columns dev_{avg} give the average deviation value of f, also computed over the same 50 validation instances. Smaller values of dev_{avg} are better; $dev = 0$ means the method reaches the best known upper bound.

On average, the learned parameters provide better solution quality, with only a slightly higher computation time. The gap in computation time increases with the instance size, but the performance improvement also becomes larger for bigger sizes.

This shows that a relationship between the instance size and the parameters exists and can be exploited to improve the algorithm's performance. The experiments with our method show that better performance can be achieved by choosing more suitable parameter values for the SACO heuristic, compared to the values manually proposed in the literature. This naturally raises the question: can performance be further improved by identifying and exploiting correlations between parameters, or between parameters and the instance of the problem?

3 Correlations between the parameters

The Bayesian optimisation framework introduced in section 2 assumes that the parameters are independent once conditioned on the instance size. However, many heuristics involve parameter roles such as exploration intensity (Nb_{ants} in the case of SACO), and computational budget (Nb_{iter}). Understanding their interaction is therefore important to reveal structure within the configuration space.

To explore these relationships, we run a grid search on SACO to evaluate a sampling of configurations (Nb_{ants}, Nb_{iter}) using the function $f(x)$. This procedure is applied to 700 randomly generated instances with sizes ranging from 20 to 150 jobs, and because SACO is stochastic, each configuration is evaluated over 10 repetitions. To each configuration (Nb_{ants}, Nb_{iter}) is associated the corresponding value of f on B_{learn}, and we only keep the ones with a value greater than or equal to 85% of the maximum of f. Figure 1 presents the corresponding configurations for instances with $n = 50$. We also indicate the configurations corresponding to the literature parameters and the ones obtained by Bayesian optimisation. To each point is shown the value of f.

Next, we identify the configurations that are Pareto optima of rank 1 and 2, in order to build an interpolation of the relation of Nb_{ants} and Nb_{iter} when looking for a performance guarantee of at least 85% of the best configurations.

We consider ranks 1 and 2 for two reasons. First, in several cases, the points on the first Pareto front are too few or too sparsely distributed to allow a reliable interpolation. Second, the points on the second front are very close in performance to those on the first front, and therefore provide additional near-optimal configurations that help obtain a more robust approximation.

We then fit a global model linking Nb_{iter}, Nb_{ants}, and the instance size n, using all selected Pareto configurations simultaneously and conditioning the relation on n. Formally, we consider a model of the form $Nb_{iter} = f(n, Nb_{ants}; x)$, where x denotes the global parameters estimated through weighted least squares. In this way, we obtain an explicit relationship describing how coupled parameter values evolve with the instance size.

The preliminary experiments on SACO indicate a clear correlation between the parameter values and the instance size. Once this process is completed and generalised, it should yield a model that adapts the parameters to the instance size while capturing possible interactions between them. This model could then be integrated into the Bayesian optimisation framework to further improve the quality of the resulting configurations.

4 Perspectives

This work shows that Bayesian optimisation can be used to outperform manual-based tuning and is a first step towards automated configuration in Operations Research. The

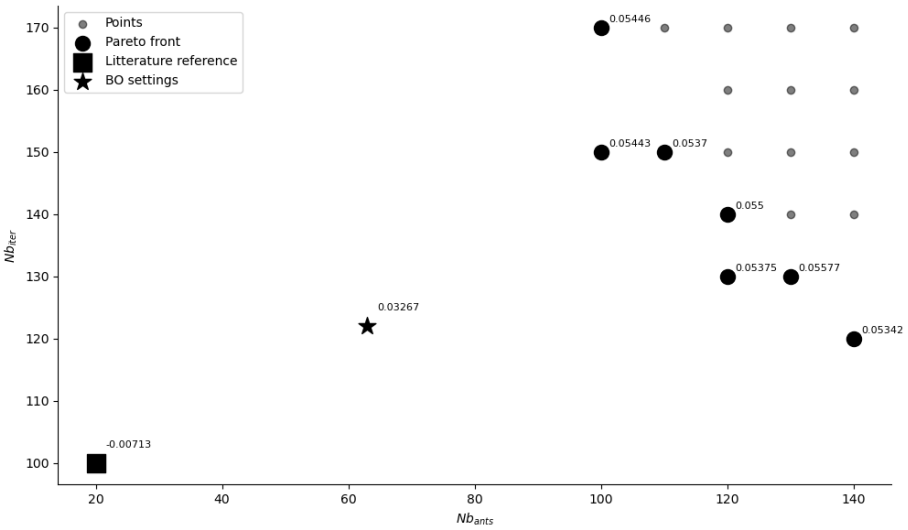


Fig. 1. Filtered high-quality SACO configurations ($\geq 85\%$) and extracted Pareto fronts for $n = 50$.

purpose is not only to improve parameter values for SACO, but also to analyse how parameter interactions evolve with instance characteristics, with a perspective of transferring this reasoning to algorithms showing comparable behaviours. We identify meaningful correlations between parameters and instance size, and future work will investigate additional instance features, such as the total processing times $\sum_{i,j} p_{j,i}$, in order to build predictive models capable of recommending parameter values tailored to each instance. Future research will aim at completing and validating such a model for the SACO heuristic, and then extending the methodology to other heuristics with similar structures or parameter roles.

References

Doerr, B. & Doerr, C. (2020). Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices, *in* B. Doerr & F. Neumann (eds), *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Springer International Publishing, pp. 271–321.

Gupta, J. N., Neppalli, V. R. & Werner, F. (2001). Minimizing total flow time in a two-machine flowshop problem with minimum makespan, *International Journal of Production Economics* **69**(3): 323–338.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly* **1**(1): 61–68.

T'kindt, V. & Billaut, J.-C. (2006). *Multicriteria Scheduling*, Springer-Verlag, Berlin/Heidelberg.

T'kindt, V., Gupta, J. N. & Billaut, J.-C. (2003). Two-machine flowshop scheduling with a secondary criterion, *Computers and Operations Research* **30**(4): 505 – 526.

T'kindt, V., Monmarché, N., Tercinet, F. & Laugt, D. (2002). An Ant Colony Optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem, *European Journal of Operational Research* **142**(2): 250–257.

Variable neighborhood descent for integrated packing and scheduling

Gustavo Alencar Rolim[1] and Silvio Alexandre de Araujo[1]

Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), Brazil
gustavo.rolim@unesp.br, silvio.araujo@unesp.br

Keywords: Packing, Scheduling, Heuristics.

1 Introduction

Classic packing problems often fail to reflect situations where packing time is as critical as the cost of unused space. Such situations frequently arise in industries that employ cutting machines, where the trade-off between productivity and efficient material utilization must be managed. Consequently, a body of literature has sought to integrate packing and scheduling decisions simultaneously, often under various assumptions regarding processing times. Recently, Marinelli, Pizzuti, Wu, and Yagiura (2025) reported an interesting problem that combines both decisions, named as the one-dimensional Bin Packing with Variable Pattern Processing Time (1BP-VPT), while their focus was on the packing perspective, the 1BP-VPT has many similarities with the Batch Scheduling Problem (BSP), which has been widely studied during the last couple of decades.

The 1BP-VPT aims to minimize a convex combination of a packing and a scheduling objective. Given weights $w_1, w_2 \geq 0$ with $w_1 + w_2 = 1$, the objective is to minimize $Z = w_1 N + w_2 \Theta$, where N is the number of bins used and Θ is a regular cost function of the completion times $C_1, C_2, \ldots, C_n$ of the n items (jobs). Each item $i \in \{1, \ldots, n\}$ has an integer length l_i, and all bins have identical integer capacity l. Using a bin requires a constant setup time $s \geq 0$, incurred before processing its assigned items and each item requires a constant processing time $t > 0$. Particular emphasis is placed on the case where $\Theta = T_{\max}$, with the due dates of item i defined as d_i, tardiness as $T_i = \max\{0, C_i - d_i\}$, and $T_{\max} = \max_{1 \leq i \leq n} T_i$. Except for a specific assumption regarding how completion times are defined, the 1BP-VPT is analogous to the bounded BSP with parallel batching (p-batch), classified as $1|\text{p-batch}, p_j = p, d_j|w_1 N + w_2 \Theta$. In the 1BP-VPT, items assigned to the same bin keep their own completion times, whereas in parallel batch scheduling all jobs in a batch share the batch completion time. Both 1BP-VPT and its BSP variant are $\mathcal{NP}$-hard problems (Marinelli et al. 2025).

Marinelli et al. (2025) introduced solution approaches for the 1BP-VPT: an exact *branch-and-price* (B&P) procedure and heuristic methods based on *sort-and-fit* (S&F) and *sequential value correction* (SVC), the latter being the current state of the art. Notably, these methods rely on well-established concepts from the bin packing and cutting stock literature, with no direct connection to the BSP. Since both problems are typically treated in isolation, this study shows that combining ideas from the two domains can yield clear benefits. We derive a *variable neighborhood descent* (VND) procedure based on neighborhood structures inspired by the BSP literature, providing an improvement phase for the SVC heuristic. The hybrid method, denoted SVC_{VND}, produces new best-known solutions (BKS) while incurring only a marginal increase in computational cost.

2 Variable neighborhood descent

The rationale for adopting a VND procedure is that a solution may be locally optimal with respect to one neighborhood but not another, whereas any global optimum must be locally optimal for all neighborhoods. VND therefore explores the solution space through multiple neighborhoods, which can be applied sequentially, in nested or composite forms, or in mixed-nested schemes (Mjirda, Todosijević, Hanafi, Hansen, and Mladenović 2016). In this study, we propose a basic VND in which the search restarts from the first neighborhood whenever the best improving solution, if any, is accepted as the new incumbent. We also introduce five neighborhood structures adapted from the intensification strategies of Queiroga, Pinheiro, Christ, Subramanian, and Pessoa (2021), with minor modifications. Algorithm 1 outlines the VND procedure. Starting from an initial solution Π, the method explores the neighborhoods in sequence. A candidate solution Π' is accepted whenever it improves the current solution, in which case the search restarts from the first neighborhood. The process continues until no further improvement is found.

- **Item insertion** ($\mathcal{N}_1$): Each item is removed from its current bin and considered for reinsertion into every other feasible bin. It may also be placed before the first bin, after the last bin, or between two consecutive bins, which allows for the creation of a singleton bin containing only that item. The neighborhood requires $n \cdot (n-1) = n^2 - n = O(n^2)$ moves, since in the worst case there can be up to n singleton bins. Evaluating the insertion of singleton bins also takes $O(n^2)$, and computing the objective function for each move requires $O(n)$. Therefore, the overall computational complexity of this neighborhood can be bounded by $O(n^3)$.
- **Item swap** ($\mathcal{N}_2$): Each item is considered for swapping with items from a different bin. A swap move is feasible only if the maximum bin length $l_{\max}$ is not exceeded in either bin. In the worst case, with n singleton bins, the neighborhood contains at most $n \cdot (n-1)/2$ candidate swaps, which can be estimated in $O(n^2)$ time. Since evaluating a single move requires $O(n)$, the overall complexity of finding the best swap move is $O(n^3)$.
- **Bin insertion** ($\mathcal{N}_3$): Each bin is removed from its current position and reinserted into every other possible position. The complexity of this neighborhood is estimated to be $O(n^3)$, since in the worst case, when all bins are singletons, the operator reduces to item insertion.
- **Bin swap** ($\mathcal{N}_4$): Each bin is swapped with a different bin. The neighborhood size is $O(n^2)$, as there are $n \cdot (n-1)/2$ possible swaps when all bins are singletons. Consequently, in the worst case, the overall complexity is $O(n^3)$, analogous to the item-swap neighborhood.
- **Bin merge** ($\mathcal{N}_5$): Two distinct bins are merged, and the resulting bin is placed at the earliest position of the two bins being merged. A merge is valid only if the resulting bin length does not exceed the maximum bin capacity. As in neighborhood $\mathcal{N}_4$, there are at most $n \cdot (n-1)/2$ possible moves. Since evaluating the objective function for each move requires $O(n)$ time, the overall complexity is $O(n^3)$.

The VND is embedded within SVC, a sequential heuristic that constructs the solution bin by bin while continuously updating *pseudo-prices*. These pseudo-prices are analogous to shadow prices in column generation, guiding the selection of promising bins by repeatedly solving a 0-1 Knapsack Problem (KP). Each KP maximizes the sum of pseudo-prices subject to the bin capacity. After a complete solution is generated, an update phase perturbs the pseudo-prices according to criteria that reflect the quality of the most recently constructed solution. We follow the scheme proposed by Marinelli et al. (2025) and set

Algorithm 1 Variable Neighborhood Descent (VND)

Require: Π, $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, |\mathcal{N}|\}$
Ensure: Π^*
$\Pi^* \leftarrow \Pi$
$k \leftarrow 1$
while $k \leq |\mathcal{N}|$ **do**
  $\Pi' \leftarrow$ `BestImprovement`$(\Pi^*, \mathcal{N}_k)$
  if $Z(\Pi') < Z(\Pi^*)$ **then**
    $\Pi^* \leftarrow \Pi'$
    $k \leftarrow 1$ ▷ Restart the search
  else
    $k \leftarrow k + 1$ ▷ Move to the next neighborhood
  end if
end while
return Π^*

SVC to terminate after 1000 iterations, than the resulting solution is further improved by VND, resulting in SVC_{VND}.

3 Results and discussion

The procedure SVC_{VND} is implemented in `C++`, compiled with `g++ 15.2.0`, and executed on a machine equipped with an Intel® Core™ i7-12700 CPU @ 2.10 GHz, 16 GB DDR4 RAM (4400 MHz), running Windows 11. Its parameters were tuned using `irace 4.2.0` and configured as $\eta_1 = 50$, $\eta_2 = 0.3$, and $\eta_3 = 1$. The results are compared with those obtained from the executable provided in the repository of Marinelli et al. (2025), also implemented in `C++`. This executable reports the best solution produced by the primal heuristics S&F[1] and the basic SVC[2]. Since this is the only benchmark available in the literature, we take the outputs of their executable as the current BKS. Specifically, we compute the relative percentage deviation (RPD) using the best objective value Z^* obtained after ten runs, each run terminated after 1000 iterations, as

$$\text{RPD} = \frac{Z^* - \text{BKS}}{\text{BKS}} \cdot 100\%.$$

The experiments are conducted on a set of instances that do not satisfy the integer round-up property, hereafter denoted `non-IRUP`[3]. This set consists of 53 instances with sizes $n \in [20, 200]$ and bin capacity $l = 1000$. According to Marinelli et al. (2025), these instances exhibit the largest average optimality gaps, especially when greater weight is assigned to the scheduling component relative to the packing term. Therefore, we set $w_1 = 0.2$, $w_2 = 0.8$, $t = 0.2$, and $s = 1$. We compare three algorithms: (i) the executable by Marinelli et al. (2025), which serves as the literature reference for BKS, (ii) SVC with parameters tuned by `irace`, denoted SVC_{P}, and (iii) SVC_{VND}.

Table 1 reports the results for the `non-IRUP` instances, grouped by number of items, presenting for each group the average RPD and the number of wins, draws, and losses (W–D–L) relative to the BKS, where negative RPD values indicate improvements and positive values indicate gaps. The aggregated results show that parameter tuning with `irace` often

[1] S&F is deterministic and runs in $O(n \log n)$.
[2] As reported by Marinelli et al. (2025), SVC is set to terminate after 1000 iterations.
[3] Due dates were added by Marinelli et al. (2025) to the packing instances of the Dresden Cutting and Packing Group (CaPaD).

Table 1. Results on `non-IRUP` instances.

Size (n)	#Inst.	SVC_P			SVC_{VND}		
		RPD	W–D–L	Time (s)	RPD	W–D–L	Time (s)
20	1	**-5.405**	1–0–0	0.039	**-5.405**	1–0–0	0.039
40	3	-4.630	1–2–0	0.128	**-5.484**	2–1–0	0.128
60	9	-4.471	9–0–0	0.306	**-4.929**	9–0–0	0.291
80	3	**-5.838**	3–0–0	0.466	**-5.838**	3–0–0	0.486
100	4	-1.709	3–1–0	0.724	**-4.180**	4–0–0	0.734
120	10	-1.241	7–3–0	1.250	**-3.041**	9–1–0	1.232
140	3	-0.406	2–0–1	1.730	**-1.627**	2–0–1	2.450
160	7	-0.670	4–0–3	2.433	**-4.820**	7–0–0	2.383
180	5	0.074	2–1–2	3.607	**-5.261**	5–0–0	3.715
200	8	0.095	4–0–4	5.079	**-6.269**	8–0–0	5.086

enhances solution quality, though not uniformly across all instance sizes; for instance, when $n \in \{180, 200\}$, the performance of SVC_P fluctuates around the BKS, balancing wins and losses and yielding a slight increase in average RPD. In contrast, SVC_{VND} improves on the BKS in all groups, with the VND-based improvement phase proving especially effective on larger instances. Despite some variation in running times across instances, Table 1 indicates that the average CPU time of SVC_{VND} increases only marginally.

4 Conclusion

In this extended abstract, we examine the connections between machine scheduling and packing, highlighting how techniques from one domain may benefit the other. To illustrate this, we develop a flexible VND framework that incorporates neighborhood structures adapted from the BSP literature. To validate our approach, we first analyze the SVC procedure proposed by Marinelli et al. (2025), currently the state of the art for solving the 1BP-VPT. We then perform additional experiments using `irace` and embedding a VND procedure as an improvement phase within SVC. The results show that the VND has the potential to improve current BKS while incurring only a marginal increase in CPU time.

Acknowledgements

We thank Dr. Andrea Pizzuti for kindly providing the executable files as well as for clarifying our inquiries. This work was supported by the Sao Paulo Research Foundation (FAPESP), Brazil [grants no. 2024/16194-3, 2024/01409-4, 2022/05803-3, 2013/07375-0] and by the National Council for Scientific and Technological Development (CNPq), Brazil [grants no. 302998/2022-5, 402240/2023-5, 402592/2024-7].

References

Marinelli, F., Pizzuti, A., Wu, W., Yagiura, M., 2025, "One-dimensional bin packing with pattern-dependent processing time", *European Journal of Operational Research*, Vol. 322, pp. 770-782.

Mjirda, A., Todosijević, R., Hanafi, S., Hansen, P., Mladenović, N., 2016, "Sequential variable neighborhood descent variants: An empirical study on the traveling salesman problem.", *International Transactions in Operational Research*, Vol. 24, pp. 615-633.

Queiroga, E., Pinheiro, R. G. S., Christ, Q., Subramanian, A., Pessoa, A. A., 2021, "Iterated local search for single machine total weighted tardiness batch scheduling", *Journal of Heuristics*, Vol. 27, pp. 353-438.

List of sponsors

Hexaly
Mathematical optimization company

ANITI
Artificial and Natural Intelligence Toulouse Institute

EURO
The association of European operational research societies

GDR ROD
CNRS Association for Operational Research and Decision Making

Université de Toulouse
University of Toulouse

INSA de Toulouse
INSA Toulouse

Springer
Springer Publishing

ROADEF
French association for operations research and decision support

LabEx CIMI
The International Center of Mathematics and Computer Science in Toulouse

OptalCP
Mathematical optimization company

Author Index