LAAS CNRS

Hackathon PMS 2026

Box Packing with Constraint Programming

Emmanuel Hebrard, Titouan Seraud
Tim Luchterhand, Mohamed Siala, Damien Piot

April 13, 2026

Hackathon Problem: Box Packing

Hackathon Problem: Box Packing

Input : A set of board games (from `https://www.espritjeu.com`)

Hackathon Problem: Box Packing

Input : A set of board games (from `https://www.espritjeu.com`)

Question : What is the smallest box (sum of length, width and height) that fits them all?

Hackathon Problem: Box Packing

Input : A set of board games (from `https://www.espritjeu.com`)

Question : What is the smallest box (sum of length, width and height) that fits them all?

- Tempo (c++20, cmake, unix of VM)
- CPMpy (Python 3.8)
- Minizinc
- Method of your choice, please form teams!

- You have until 18:00 to submit solutions (positition of each box); rotations are allowed (if axis-aligned)

Scheduling VS. (Rectangle/3D) Packing

- Scheduling problem = *precedence relations* + *resource constraints* + *objective function*

Rectangle packing

- Consider the X axis: a rectangle is a task/interval of duration equal to its length
- It takes up resources: space on the Y axis

Modeling a scheduling problem with CP: *sequencing* or *timetabling*?

- *Timetabling*: assigning a timepoint to events
- *Sequencing*: deciding of the order between events

Modeling a scheduling problem with CP: *sequencing* or *timetabling*?

- *Timetabling*: assigning a timepoint to events
- *Sequencing*: deciding of the order between events

Sequencing is usually better than timetabling

Modeling a scheduling problem with CP: *sequencing* or *timetabling*?

- *Timetabling*: assigning a timepoint to events
- *Sequencing*: deciding of the order between events

Sequencing is usually better than timetabling

- Complexity does not depend on the time (space) scale

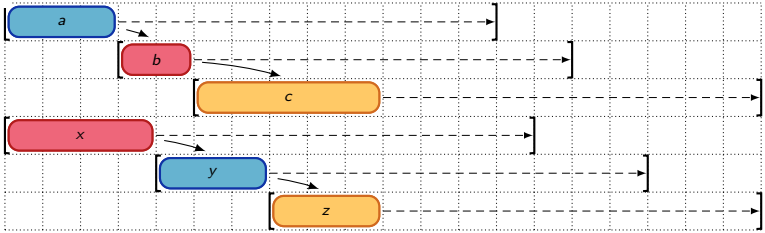
Modeling a scheduling problem with CP: *sequencing* or *timetabling*?

- *Timetabling*: assigning a timepoint to events
- *Sequencing*: deciding of the order between events

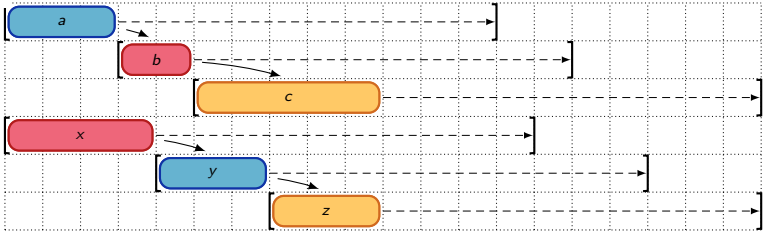
Sequencing is usually better than timetabling

- Complexity does not depend on the time (space) scale
- Search space is smaller: distinct sequences implies distinct timetables, but the converse is not true

- Example: jobshop scheduling (minimum makespan)

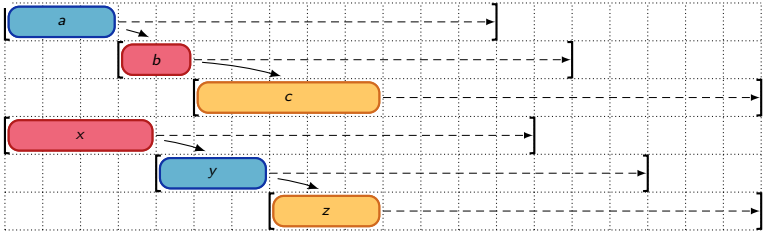


- Example: jobshop scheduling (minimum makespan)




```
Model model;

auto sa = model.newNumeric("start_a");
auto sb = model.newNumeric("start_b");
auto sc = model.newNumeric("start_c");

auto sx = model.newNumeric("start_x");
auto sy = model.newNumeric("start_y");
auto sz = model.newNumeric("start_z");
```

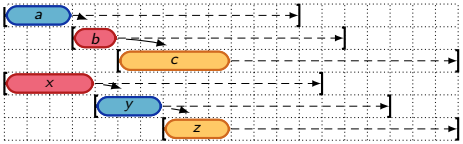
- Example: jobshop scheduling (minimum makespan)



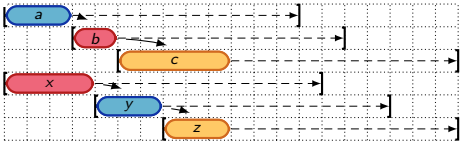

```
Model model;

auto sa = model.newNumeric("start_a");
auto sb = model.newNumeric("start_b");
auto sc = model.newNumeric("start_c");

auto sx = model.newNumeric("start_x");
auto sy = model.newNumeric("start_y");
auto sz = model.newNumeric("start_z");
```

```
auto interval_a = model.between(sa, sa+3);
auto interval_b = model.between(sb, sb+2);
auto interval_c = model.between(sc, sc+5);

auto interval_x = model.between(sx, sx+4);
auto interval_y = model.between(sy, sy+3);
auto interval_z = model.between(sz, sz+3);

auto makespan = model.newNumeric("makespan");
```

Posting Constraints

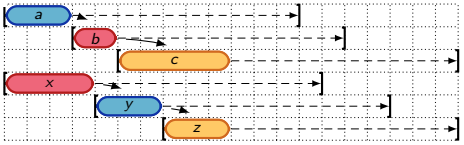


Posting Constraints



Precedences

```
model.post(0 <= interval_a.start);
model.post(interval_a.end <= interval_b.start);
model.post(interval_b.end <= interval_c.start);
model.post(interval_c.end <= makespan);

model.post(0 <= interval_x.start);
model.post(interval_x.end <= interval_y.start);
model.post(interval_y.end <= interval_z.start);
model.post(interval_z.end <= makespan);
```

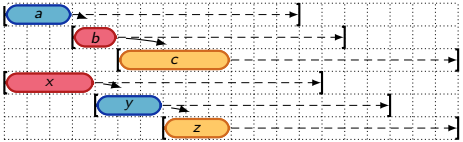


Precedences

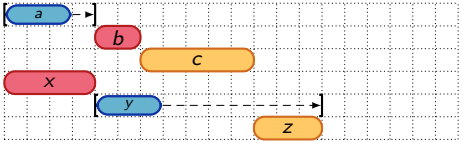
```
model.post(0 <= interval_a.start);
model.post(interval_a.end <= interval_b.start);
model.post(interval_b.end <= interval_c.start);
model.post(interval_c.end <= makespan);

model.post(0 <= interval_x.start);
model.post(interval_x.end <= interval_y.start);
model.post(interval_y.end <= interval_z.start);
model.post(interval_z.end <= makespan);
```

Resources

```
auto a_before_y = model.existBoolean("a < y");
model.post(a_before_y <=> (interval_a.end <= interval_y.start));
auto y_before_a = model.existBoolean("y < a");
model.post(y_before_a <=> (interval_y.end <= interval_a.start));
model.post(a_before_y or y_before_a);

auto b_before_x = model.existBoolean("b < x");
model.post(b_before_x <=> (interval_b.end <= interval_x.start));
auto x_before_b = model.existBoolean("x < b");
model.post(x_before_b <=> (interval_x.end <= interval_b.start));
model.post(b_before_x or x_before_b);

auto c_before_z = model.existBoolean("c < z");
model.post(c_before_z <=> (interval_c.end <= interval_z.start));
auto z_before_c = model.existBoolean("z < c");
model.post(z_before_c <=> (interval_z.end <= interval_c.start));
model.post(c_before_z or z_before_c);
```




```
Solver solver(model);

auto objective = Minimize(makespan);

auto result = solver.optimize(objective);

if(result != Outcome::Unsatisfiable) {
    Solution solution(solver);
    std::cout << "a: " << solution(sa) << " b: " << solution(sb) << " c: " << solution(sc) << std::endl
    << "x: " << solution(sx) << " y: " << solution(sy) << " z: " << solution(sz) << std::endl;
}
```




```
Solver solver(model);

auto objective = Minimize(makespan);

auto result = solver.optimize(objective);

if(result != Outcome::Unsatisfiable) {
    Solution solution(solver);
    std::cout << "a: " << solution(sa) << " b: " << solution(sb) << " c: " << solution(sc) << std::endl
    << "x: " << solution(sx) << " y: " << solution(sy) << " z: " << solution(sz) << std::endl;
}
```

```
a: [0..1] b: 4 c: 6
x: 0 y: [4..8] z: 11
```

Decision or Auxiliary Variables

```
Model model;

auto x = model.existNumeric(); // decision variable
auto y = model.newNumeric(); // auxiliary variable

auto a = model.existBoolean(); // decision variable
auto b = model.newBoolean(); // auxiliary variable
...
Solver solver(model);
if(solver.satisfiable()) {
    Solution solution(solver);
}
```

- `solution(x)` and `solution(y)` are ranges, but `solution(x).min` $=$ `solution(x).max` whereas `solution(y).min` $\leq$ `solution(y).max`
- `solution(a)` $\in$ {`TruthVal::True`, `TruthVal::False`}
- `solution(b)` $\in$ {`TruthVal::True`, `TruthVal::False`, `TruthVal::Undefined`}
- `solution[x]`, `solution[y]`, `solution[a]`, `solution[b]`, return the expected type (`int` or `bool`) but may raise an exception

When can care only about the sequence?

When can care only about the sequence?

- Almost always!

When can care only about the sequence?

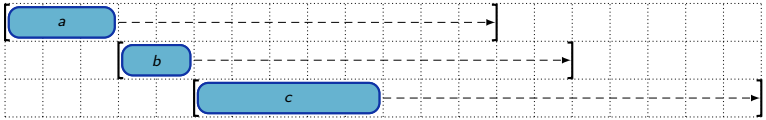
- Almost always!
- Scheduling problem = *precedence relations* + *resource constraints* + *objective function*

When can care only about the sequence?

- Almost always!
- Scheduling problem = *precedence relations* + *resource constraints* + *objective function*

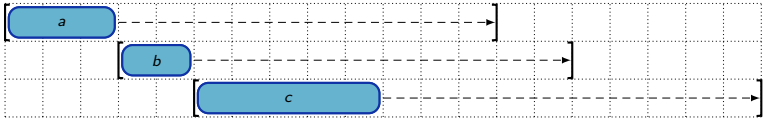
Base case : difference logic systems / simple temporal networks / PERT diagrams...

- Temporal/spatial variables (numeric) with precedences: $x + k \leq y$ with x, y two variables and k a constant
- Solved by simple *constraint propagation* (polynomial) and bounds are consistent *simultaneously*
  - assigning every variable to its lower (resp. upper) bound is a solution



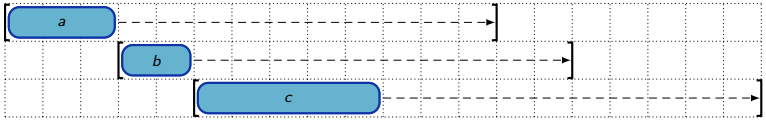
Disjunctive resources

- Can be written as a disjunction of precedences, e.g., $([s_a + 3 \leq s_b]$ or $[s_b + 2 \leq s_a])$ and...



Disjunctive resources

- Can be written as a disjunction of precedences, e.g., $([s_a + 3 \leq s_b]$ or $[s_b + 2 \leq s_a])$ and...
- Once at least one disjunct is true in each disjunction, no solution feasible for this set of precedences may violate the resource constraints

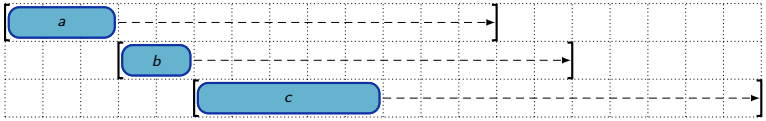


Disjunctive resources

- Can be written as a disjunction of precedences, e.g., $([s_a + 3 \leq s_b]$ or $[s_b + 2 \leq s_a])$ and...
- Once at least one disjunct is true in each disjunction, no solution feasible for this set of precedences may violate the resource constraints

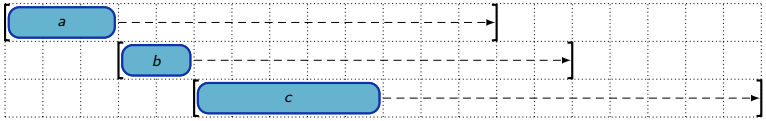
Cumulative resources

- Not as straightforward, but there are efficient algorithms to obtain the same property



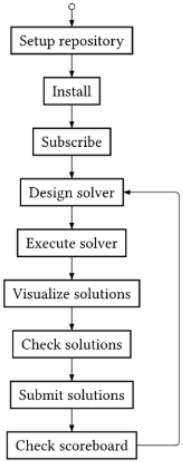
- Many objective functions are monotone: "the earlier the better"
  - Minimum makespan, minimum sum of tardiness, etc.

Objective Function



- Many objective functions are monotone: "the earlier the better"
  - Minimum makespan, minimum sum of tardiness, etc.
- The solution where every event goes to its lower bound is the best!

- Go to `https://pms2026.sciencesconf.org/` then Master Class then Hackathon Results and Hackathon Instructions
- Install the python helper scripts: `pip install git+https://gitlab.laas.fr/roc/titouan-seraud/pms.git`
- Clone the git repo of the hackathon, make your own copy repo, and send it by email to `titouan.seraud@laas.fr`
  - Fill the file `team.json` with your names
  - Folder `dataset` contains the instances
  - Copy solutions in folder `solutions` and push
- Empty models, parser and solution writting for `Minizinc`, `CPMpy` and `Tempo`



Go!

- First, try simply to model the problem

Go!

- First, try simply to model the problem
- Likely not very efficient, probably not because of the model

Go!

- First, try simply to model the problem
- Likely not very efficient, probably not because of the model
  - Solving with different parameters
  - Providing initial solutions
  - User-defined search
  - Large Neighborhood Search
  - Easy tricks, be creative!