RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

Basics of Constraint Programming for Scheduling and Application to Space

Cédric Pralet (ONERA/DTIS)
PMS, April 13th, 2026

Preliminary remarks

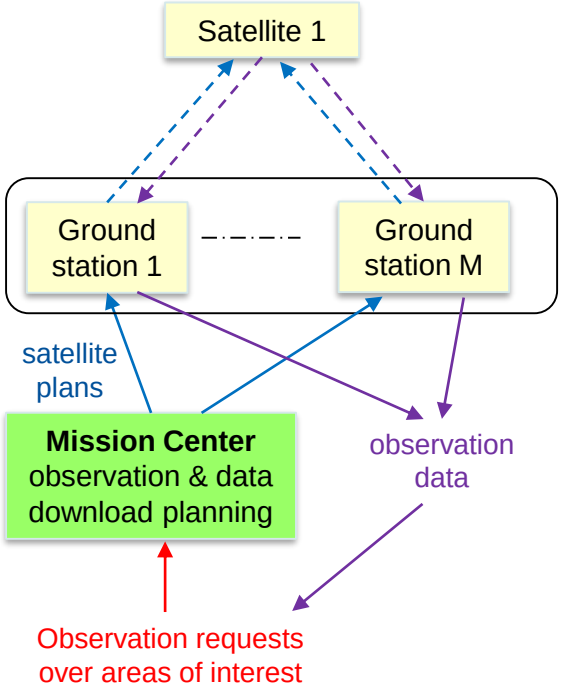
Background

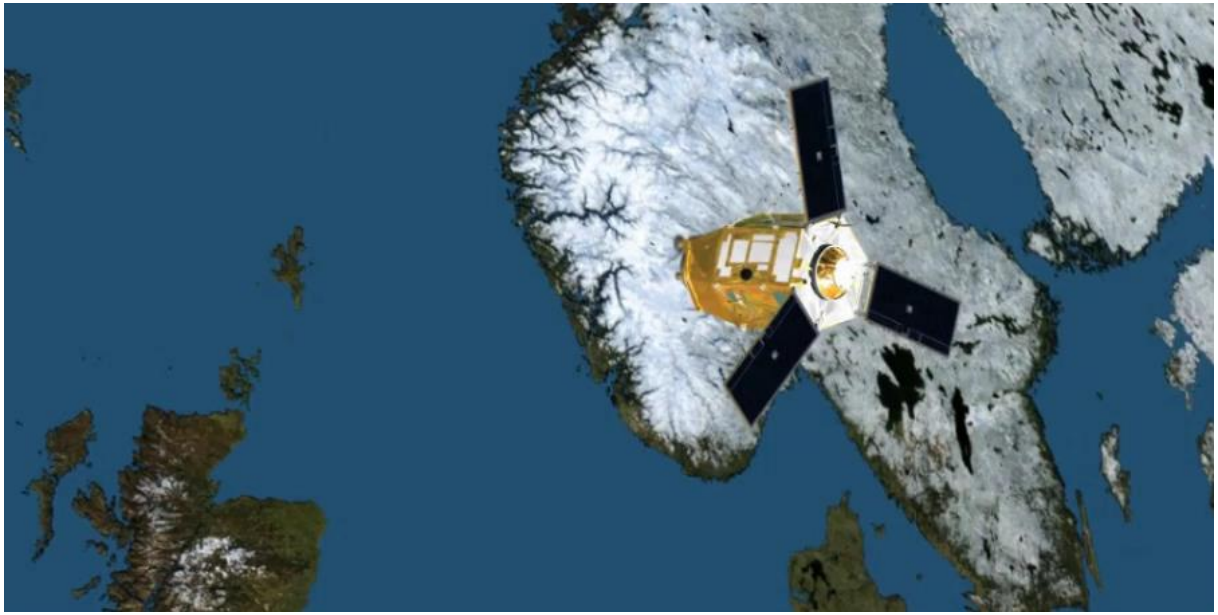
- Optimization in the space domain
- Studies with academic / institutional / industrial partners (CNES, Airbus DS, ESA, Thales Alenia Space, Prométhée Earth Intelligence, LAAS-CNRS…)
- Usage of several Operations Research methods : greedy search, local search, metaheuristics, SAT, MILP, Constraint Programming (CP), hybrid methods…
- Developer of metaheuristic solutions (ILS, LNS…), constraint-based local search…

Disclaimers

- Personally not a core CP engine developer (user perspective)
- Only two solvers considered here : CPOptimizer (IBM) and OR-Tools CP-SAT (Google)
- A single application : planning for Earth Observation Satellites (EOS)

EOS planning





Organization

1. Constraint Programming (CP) : generalities

Constraint Satisfaction Problem

Constraint Satisfaction Problem (CSP) = triple (X,D,C)

- $X = \{x_1, \dots, x_n\}$: finite set of variables
- D = function defining the domain of each variable ($D(x_i)$ = possible values for $x_i \in X$)
- $C = \{c_1, \dots, c_m\}$: finite set of constraints

Constraint c = pair (S, R) where

- $S = [y_1, \dots, y_k]$: list of variables $y_i \in X$ over which the constraint holds
- $R \subseteq D(y_1) \times \cdots \times D(y_k)$: allowed combinations of values for the variables in S

Different ways to define a constraint

- List of accepted or forbidden combination of values (ex : $(x, y, z) \in \{(0,1,2), (0,2,1), (1,2,0)\}$)
- Usage of arithmetic or logical operators (ex : $(x < y)\ AND\ (x \neq z)\ and\ (y \neq z)$)
- Usage of constraints available in a catalog (ex: $(x \neq y)\ AND\ alldifferent(x, y, z)$)
- Specific code

Constraint Satisfaction Problem

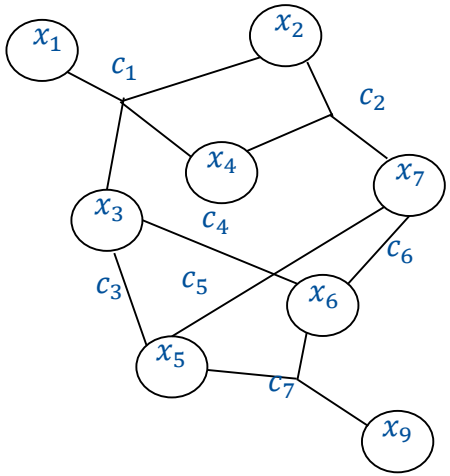
Graphical representation : constraint network

Assignment A

- function assigning a value $A(x) \in D(x)$ to each variable $x \in X$

Solution

- assignment A such that all the constraints in C are satisfied
- Formally, $\forall c = (S, R) \in C$ where $S = [y_1, \dots, y_k]$, condition « $(A(y_1), \dots, A(y_k)) \in R$ » holds



COP (Constraint Optimization Problem)

- addition of an objective to optimize
- equivalent to the resolution of a sequence of satisfaction problems

Constraint propagation to prune inconsistent values

Examples

- domains $D(x) = [0..5], D(y) = [2..6]$, constraint c : $x \geq y$
  possibility to reduce the domains by $D'(x) = [2..5], D'(y) = [2..5]$
- domains $D(x) = [0..2], D(y) = [0..2]$, constraint c : y = tab[x] where $tab = [1,3,0,2,1]$
  possibly to reduce the domains by $D'(x) = \{0,2\}, D'(y) = \{0,1\}$ (element constraint)

Tuned constraint propagation algorithm for each constraint type, establishing different kinds of consistency properties (arc consistency (AC), generalized arc consistency (GAC), bound consistency…)

Concept of propagation

- pruning events triggered by one constraint may induce new pruning opportunities for other constraints
- pruning until a fixed point is reached
- inconsistency detection (i.e., no solution) if the domain of a variable becomes empty

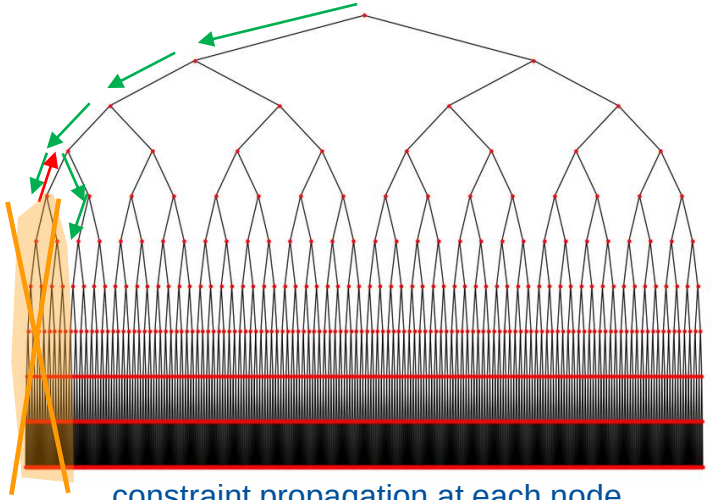
Propagation not sufficient to find a solution or detect inconsistency in general (incomplete mechanism)

Tree search with constraint propagation

Basic search strategy : depth-first search with backtracking

At each search node

- Select a variable x
  (var-choice heuristic, e.g. mindom, mindom/deg, wdeg…)
- Select a value or a domain splitting strategy
- Branch (e.g., x = a versus x != a) and explore a child node
- Propagate the constraints at the child node
- Backtrack in case of inconsistency (empty var domain)



constraint propagation at each node

Stop when solution found or backtrack from the root note (no solution)

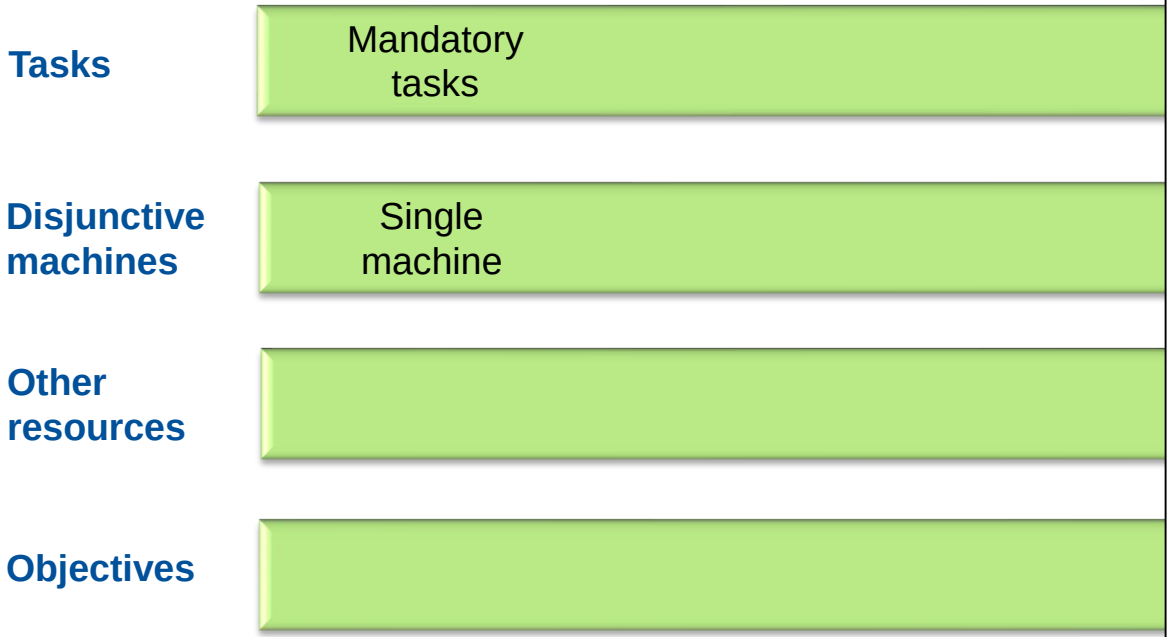
Many other methods : enhanced propagation at the root node, clause learning, backjumping, restarts, conflict-based heuristics, symmetry breaking… => NOT DISCUSSED HERE

Scheduling problems and CP

Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap

Scheduling problems and CP



Problem

- one satellite over one orbit
- set of observation tasks
- time windows (release dates and deadlines)
- fixed transition times between tasks (integrated into the task duration)

Problem definition

N observation tasks, with for each $i \in [1..N]$:

- $EST[i]$: earliest start time for obs i
- $LST[i]$: latest start time for obs i
- $DU[i]$: duration during which i uses the satellite (obs. duration + fixed transition time to next obs)

Constraints

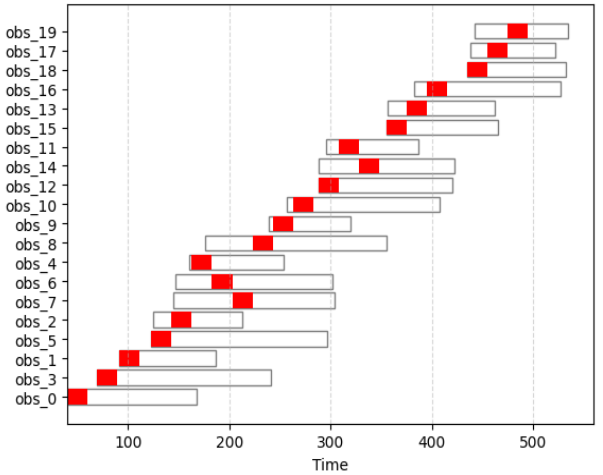
- Observations performed within the windows
- No overlapping between the observations

Objective : perform all the observations

Small note : for some models, addition of a fictitious obs 0


```
tasks:
  0:
    name:           "obs_0"
    earliestStart:  40
    latestStart:    148
    duration:       20
  1:
    name:           "obs_1"
    earliestStart:  91
    latestStart:    167
    duration:       20
  2:
    name:           "obs_2"
    earliestStart:  125
    latestStart:    193
    duration:       20
  3:
    name:           "obs_3"
    earliestStart:  69
    latestStart:    221
    duration:       20
```


Solution example



Some possible CP models

Variables

- $start[i] \in [EST[i]..LST[i]]$ = start time of obs i

Constraints to ensure non-overlapping

- $\forall i,j \in [1..N]\ s.t.\ i < j$:
  $(start[i] + DU[i] \leq start[j])$
  $OR(start[j] + DU[j] \leq start[i])$

M1

Variables

- $start[i] \in [EST[i]..LST[i]]$ = start time of obs i
- $nextId[i] \in [0..N]$ = index of the obs following i in the sequence of tasks

Constraints to ensure non-overlapping

- $\forall i \in [1..N]$:
  $start[nextId[i]] \geq start[i] + DU[i]$
- $allDifferent(\{nextId[i] \mid i \in [0..N]\})$

M2

Variables

- $start[i] \in [EST[i]..LST[i]]$ = start time of obs i
- $next[i,j] \in \{0,1\}$ = obs j just after obs i (for $i \neq j$)

Constraints to ensure non-overlapping

- $\forall i,j \in [1..N]\ s.t.\ i \neq j$:
  $(next[i,j] = 1) \rightarrow (start[j] \geq start[i] + DU[i])$
- $circuit(\{(i,j,next[i,j]) \mid i,j \in [0..N], i \neq j\})$

M3

Variables

- Interval $itv[i]$ of size $DU[i]$ in window $[EST[i]..LST[i] + DU[i]]$

Constraint to ensure non-overlapping

- $noOverlap(\{itv[i] \mid i \in [1..N]\})$

Very elegant + efficient propagators for the noOverlap constraint (overload checking, edge-finding…)

M4

Some possible CP models : solver usage

M4, CPOptimizer (Java API)

```
IloCP cp = new IloCP();
IloIntervalVar[] obsItvs = new IloIntervalVar[nTasks];
for(int i=0;i<nTasks;i++) {
    Task task = pb.tasks.get(i);
    IloIntervalVar itv = cp.intervalVar(task.duration);
    itv.setStartMin(task.earliestStart);
    itv.setStartMax(task.latestStart);
    obsItvs[i] = itv;
}
cp.add(cp.noOverlap(obsItvs));
```

M4, OR-Tools CP-SAT (Python API)

```
model = cp_model.CpModel()
intervals = []
for task in tasks:
    (name,est,lst,du) = (task['name'], task['earliestStart'],
                         task['latestStart'], task['duration'])
    start = model.NewIntVar(est, lst, f"start({name})")
    end = model.NewIntVar(est + du, lst + du, f"end({name})")
    interval = model.NewIntervalVar(start, du, end, f"itv({name})")
    intervals.append(interval)

model.AddNoOverlap(intervals)
```

Comparison for 100 observation tasks

M1 (time + or)

```
! ---------------------------------------------------- CP Optimizer 22.1.1.0 --
! Satisfiability problem - 100 variables, 4 950 constraints
! Initial process time : 0,05s (0,05s extraction + 0,00s propagation)
!  . Log search space  : 654,8 (before), 654,8 (after)
!  . Memory usage      : 626,6 kB (before), 626,6 kB (after)
! Using parallel search with 20 workers.
! ----------------------------------------------------------------------------
!              Branches  Non-fixed    W       Branch decision
                 1 000         37    1   F   892  = _int37
                 1 000         32    2   F   728  = _int31
                 1 000         51    3   F   515  = _int20
                 1 000         37    4   F 1 283  = _int60
                 1 000         45    5   F   350  = _int12
                 1 000         43    6   F 1 892  = _int92
                 1 000         40    7       658 != _int25
                 1 000         33    8   F   753  = _int32
                 1 000         41    9     1 298 != _int59
*                  117  0,08s       10       111  = _int0
! ----------------------------------------------------------------------------
! Search completed, 1 solution found.
! ----------------------------------------------------------------------------
! Number of branches     : 36 382
! Number of fails        : 16 244
! Total memory usage     : 39,7 MB (34,3 MB CP Optimizer + 5,4 MB Concert)
! Time spent in solve    : 0,09s (0,04s engine + 0,05s extraction)
! Search speed (br. / s) : 909 550,0
! ----------------------------------------------------------------------------
```

M4 (itv + noOverlap)

```
! ---------------------------------------------------- CP Optimizer 22.1.1.0 --
! Maximization problem - 101 variables, 1 constraint
! Initial process time : 0,01s (0,01s extraction + 0,00s propagation)
!  . Log search space  : 636,1 (before), 636,1 (after)
!  . Memory usage      : 587,7 kB (before), 587,7 kB (after)
! Using parallel search with 20 workers.
! ----------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                       0        101                    -
+ New bound is 0
! Using iterative diving.
*            0      101  0,05s        1      (gap is 0,00%)
! ----------------------------------------------------------------------------
! Search completed, 1 solution found.
! Best objective         : 0 (optimal - effective tol. is 0)
! Best bound             : 0
! ----------------------------------------------------------------------------
! Number of branches     : 185 070
! Number of fails          5 671
! Total memory usage       10,8 MB (10,7 MB CP Optimizer + 0,1 MB Concert)
! Time spent in solve      0,05s (0,04s engine + 0,01s extraction)
! Search speed (br. / s) : 4 626 750,1
! ----------------------------------------------------------------------------
```

Comparison for 700 observation tasks

M1 (time + or)

```
! ----------------------------------------------------- CP Optimizer 22.1.1.0 --
! Satisfiability problem - 700 variables, 244 650 constraints
! Initial process time : 2,34s (2,33s extraction + 0,01s propagation)
!  . Log search space  : 4 531,1 (before), 4 531,1 (after)
!  . Memory usage      : 11,5 MB (before), 11,5 MB (after)
! Using parallel search with 20 workers.
! ----------------------------------------------------------------------------
!              Branches  Non-fixed    W       Branch decision
                 1 000        483    1     8 594  = _int424
                 1 000        486    2   F 11 599  = _int577
                 1 000        462    3   F 1 124  = _int51
                 1 000        482    4     3 848 != _int186
                 1 000        399    5     9 842 != _int483
                 1 000        455    6   F 5 811  = _int285
                 1 000        443    7   F 6 146  = _int303
                 1 000        647    8     2 477 != _int119
                 1 000        562    9     5 213  = _int259
                 1 000        373   10   F 3 992  = _int194
                 1 000        529   11     10 025  = _int497
                 1 000        650   12     9 919 != _int490
                 1 000        396   13     7 861 != _int391
                 1 000        357   14     1 633  = _int77
                 1 000        480   15   F 1 685  = _int75
*                  933  2,79s       16     14 097  = _int699
! ----------------------------------------------------------------------------
! Search completed, 1 solution found.
! ----------------------------------------------------------------------------
! Number of branches     : 38 982
! Number of fails        : 14 869
! Total memory usage     : 942,2 MB (686,4 MB CP Optimizer + 255,8 MB Concert)
! Time spent in solve    : 2,83s (0,50s engine + 2,33s extraction)
! Search speed (br. / s) : 77 984,0
! ----------------------------------------------------------------------------
```

M4 (itv + noOverlap)

```
! ----------------------------------------------------- CP Optimizer 22.1.1.0 --
! Maximization problem - 701 variables, 1 constraint
! Initial process time : 0,01s (0,01s extraction + 0,00s propagation)
!  . Log search space  : 4 534,9 (before), 4 534,9 (after)
!  . Memory usage      : 2,0 MB (before), 2,0 MB (after)
! Using parallel search with 20 workers.
! ----------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                       0        701                    -
+ New bound is 0
! Using iterative diving.
*            0      691  0,07s        1      (gap is 0,00%)
! ----------------------------------------------------------------------------
! Search completed, 1 solution found.
! Best objective         : 0 (optimal - effective tol. is 0)
! Best bound             : 0
! ----------------------------------------------------------------------------
! Number of branches     : 79 308
! Number of fails        : 1 144
! Total memory usage     : 35,3 MB (34,6 MB CP Optimizer + 0,7 MB Concert)
! Time spent in solve    : 0,09s (0,08s engine + 0,01s extraction)
! Search speed (br. / s) : 1 132 971,4
! ----------------------------------------------------------------------------
```

Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks

Scheduling problems and CP

Tasks

Mandatory tasks — Optional tasks

Disjunctive machines

Single machine

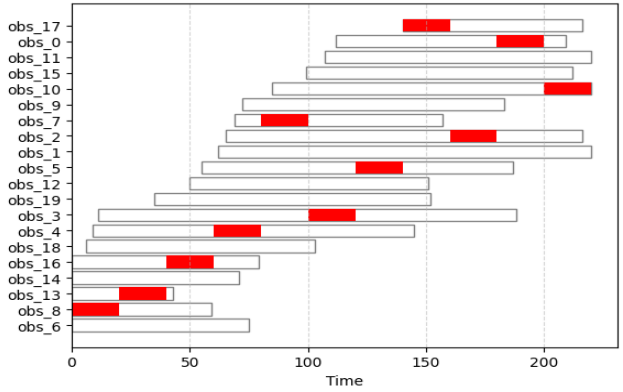
Other resources

Objectives

Rewards or priorities

Problem

- one satellite over one orbit
- set of possible observation tasks
- time windows
- fixed transition times between tasks
- reward R[i] for obs. i, depending on the cloud cover forecast



CP models

M1 with optional obs

Variables

- $start[i] \in [EST[i]..LST[i]]$ = start time of obs i
- $pres[i] \in \{0,1\}$ = selection of obs i

Constraints

- $\forall i,j \in [1..N]\ s.t.\ i < j$:
  $(pres[i] = 1\ AND\ pres[j] = 1) \rightarrow (start[i] + DU[i] \leq start[j]\ OR\ start[j] + DU[j]\ \leq start[i])$

Objective : $maximize\ sum(i \in [1..N])\ pres[i] * R[i]$

M4 with optional obs

Variables

- Interval $itv[i]$ of size $DU[i]$ in window $[EST[i]..LST[i] + DU[i]]$ optional

Constraint to ensure non-overlapping

- $noOverlap(\{itv[i] \mid i \in [1..N]\})$

Objective : $maximize\ sum(i \in [1..N])\ presence(itv[i]) * R[i]$

Still a very compact declarative model

Some possible CP models : solver usage

M4, CPOptimizer (Java API)

```
IloCP cp = new IloCP();
IloIntervalVar[] obsItvs = new IloIntervalVar[nTasks];
for(int i=0;i<nTasks;i++) {
    Task task = pb.tasks.get(i);
    IloIntervalVar itv = cp.intervalVar(task.duration);
    itv.setStartMin(task.earliestStart);
    itv.setStartMax(task.latestStart);
    itv.setOptional();
    obsItvs[i] = itv;
}
cp.add(cp.noOverlap(obsItvs));

IloIntExpr totalReward = cp.intExpr();
for(int i=0;i<nTasks;i++) {
    totalReward = cp.sum(totalReward,
            cp.prod(cp.presenceOf(obsItvs[i]), pb.tasks.get(i).reward));
}
cp.addMaximize(totalReward);
```

M4, OR-Tools CP-SAT (Python API)

```
presences = []
intervals = []
for task in tasks:
    (name,est,lst,du) = (task['name'], task['earliestStart'],
                         task['latestStart'], task['duration'])
    start = model.NewIntVar(est, lst, f"start({name})")
    end = model.NewIntVar(est + du, lst + du, f"end({name})")
    presence = model.NewBoolVar(f"presence({name})")
    interval = model.NewOptionalIntervalVar(start, du, end, presence, f"itv({name})")
    presences.append(presence)
    intervals.append(interval)

model.AddNoOverlap(intervals)

model.Maximize(sum(tasks[i]['reward'] * presences[i] for i in range(len(tasks))))
```

Model comparison

M1 (time + or), 20 obs tasks

M4 (itv + noOverlap), 20 obs. tasks

```
! ---------------------------------------------------- CP Optimizer 22.1.1.0 --
! Maximization problem - 21 variables, 1 constraint
! Initial process time : 0,00s (0,00s extraction + 0,00s propagation)
!  . Log search space  : 87,8 (before), 87,8 (after)
!  . Memory usage      : 443,4 kB (before), 443,4 kB (after)
! Using parallel search with 20 workers.
! ---------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
                       0         21                 -
+ New bound is 44
! Using iterative diving.
! Using temporal relaxation.
                       0         20    1            -
+ New bound is 31
*            26       27  0,06s        1      (gap is 19,23%)
*            28      182  0,06s        1      (gap is 10,71%)
*            31      240  0,06s        1      (gap is 0,00%)
! ---------------------------------------------------------------------------
! Search completed, 3 solutions found.
! Best objective         : 31 (optimal - effective tol. is 0)
! Best bound             : 31
! ---------------------------------------------------------------------------
! Number of branches     : 180 906
! Number of fails        : 14 407
! Total memory usage     : 8,3 MB (8,2 MB CP Optimizer + 0,1 MB Concert)
! Time spent in solve    : 0,06s (0,06s engine + 0,00s extraction)
! Search speed (br. / s) : 3 618 119,9
! ---------------------------------------------------------------------------
Solution found 31.0
```

100 tasks : search terminated in 0.42s
500 tasks : high-quality solutions in a few seconds

Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times

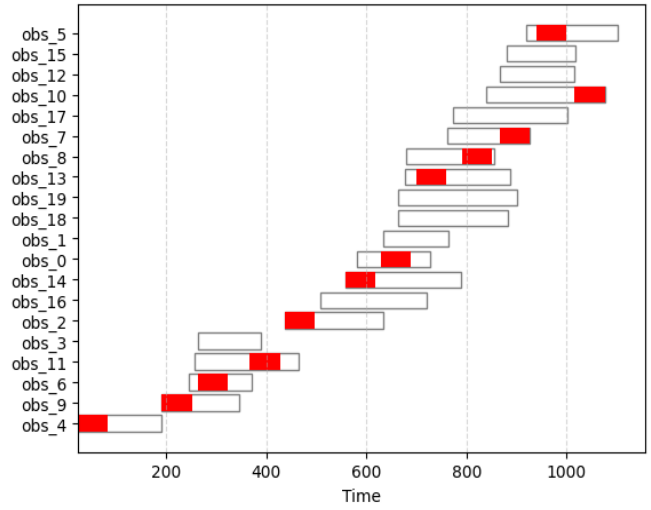
Scheduling problems and CP

Tasks	Mandatory tasks	Optional tasks
Disjunctive machines	Single machine	Sequence-dependent setup times
Other resources		
Objectives	Rewards or priorities	

New specification

- sequence-dependent transition times between the tasks

```
transitionTimes:
[[ 0 16 30  7 13  9 13 22 33  5]
 [16  0 44 24 12 20 12 31 44 10]
 [30 44  0 25 43 24 43 18  8 34]
 [ 7 24 25  0 18 11 18 22 29 13]
 [13 12 43 18  0 21  0 34 46 11]
 [ 9 20 24 11 21  0 21 13 25 11]
 [13 12 43 18  0 21  0 34 46 11]
 [22 31 18 22 34 13 34  0 15 23]
 [33 44  8 29 46 25 46 15  0 36]
 [ 5 10 34 13 11 11 11 23 36  0]]
```



NoOverlap with setup times

Interval-based model, CPOptimizer version

- Concept of sequence variable
- Concept of interval type (in our case, interval type = observation id)
- Possibility to define a noOverlap constraint with setup times over a sequence variable
- TT[k,k'] = setup time required between a task of type k and a task of type k'

```
IloCP cp = new IloCP();
int nTasks = pb.tasks.size();
IloIntervalVar[] obsItvs = new IloIntervalVar[nTasks];
for(int i=0;i<nTasks;i++) {
    Task task = pb.tasks.get(i);
    IloIntervalVar itv = cp.intervalVar(task.duration);
    itv.setStartMin(task.earliestStart);
    itv.setStartMax(task.latestStart);
    obsItvs[i] = itv;
    itv.setOptional();
}

int[] types = new int[nTasks];
for(int i=0;i<nTasks;i++) types[i] = i;
IloIntervalSequenceVar globalSequence = cp.intervalSequenceVar(obsItvs, types);
IloTransitionDistance tdist = cp.transitionDistance(pb.setupTimes);
cp.add(cp.noOverlap(globalSequence, tdist));

IloIntExpr totalReward = cp.intExpr();
for(int i=0;i<nTasks;i++) {
    totalReward = cp.sum(totalReward,
            cp.prod(cp.presenceOf(obsItvs[i]), pb.tasks.get(i).reward));
}
cp.addMaximize(totalReward);
```

NoOverlap with setup times

OR-Tools CP-SAT :

- back to a circuit constraint to enforce the setup times (routing constraint)
- combined with a standard noOverlap (scheduling constraint)
- somehow a more low-level description compared to CPOptimizer

```
for task in tasks:
    (name,est,lst,du) = (task['name'], task['earliestStart'],
                         task['latestStart'], task['duration'])
    start = model.NewIntVar(est, lst, f"start({name})")
    end = model.NewIntVar(est + du, lst + du, f"end({name})")
    presence = model.NewBoolVar(f"presence({name})")
    interval = model.NewOptionalIntervalVar(start, du, end, presence, f"itv({name})")
    starts.append(start)
    ends.append(end)
    presences.append(presence)
    intervals.append(interval)

model.AddNoOverlap(intervals)
```

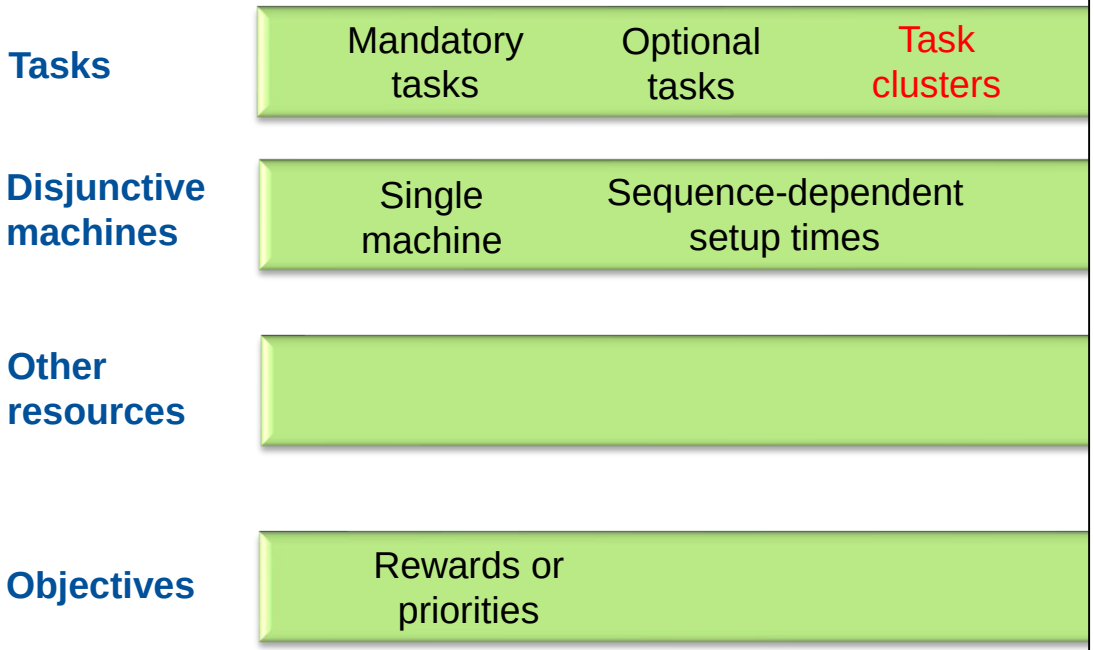
```
arcs = []
for i in range(nTasks):
    first = model.NewBoolVar(f"first_{i}")
    arcs.append((0, i+1, first))
    model.AddImplication(first, presences[i])
    last = model.NewBoolVar(f"last_{i}")
    arcs.append((i+1, 0, last))
    model.AddImplication(last, presences[i])

for i in range(nTasks):
    arcs.append((i+1, i+1, presences[i].Not()))
    for j in range(nTasks):
        if i != j:
            nextij = model.NewBoolVar(f"next_{i}_{j}")
            arcs.append((i+1, j+1, nextij))
            model.AddImplication(nextij, presences[i])
            model.AddImplication(nextij, presences[j])
            model.Add(starts[j] >= ends[i] + setupTime[i][j]).OnlyEnforceIf(nextij)

model.AddCircuit(arcs)
```

```
model.Maximize(sum(tasks[i]['reward'] * presences[i] for i in range(nTasks)))
```

Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times
5. Scheduling with CP : task clusters

Scheduling problems and CP



Problem

- one satellite over one orbit
- set of possible observation tasks
- time windows
- fixed transition times between tasks
- task clusters : one cluster = a group of tasks that must all be performed to get a reward
  (ex : stereo requests, synchronized multi-site observation requests…)
- reward Reward[c] for cluster c

Task clusters : easy in CP

Additional input data

- $Clusters$: set of task clusters, with for each cluster c a set of observation indices $I(c) \subseteq [1..N]$

Additional variables

- $satisfied[c] \in \{0,1\}$: cluster c satisfied or not

Additional constraints

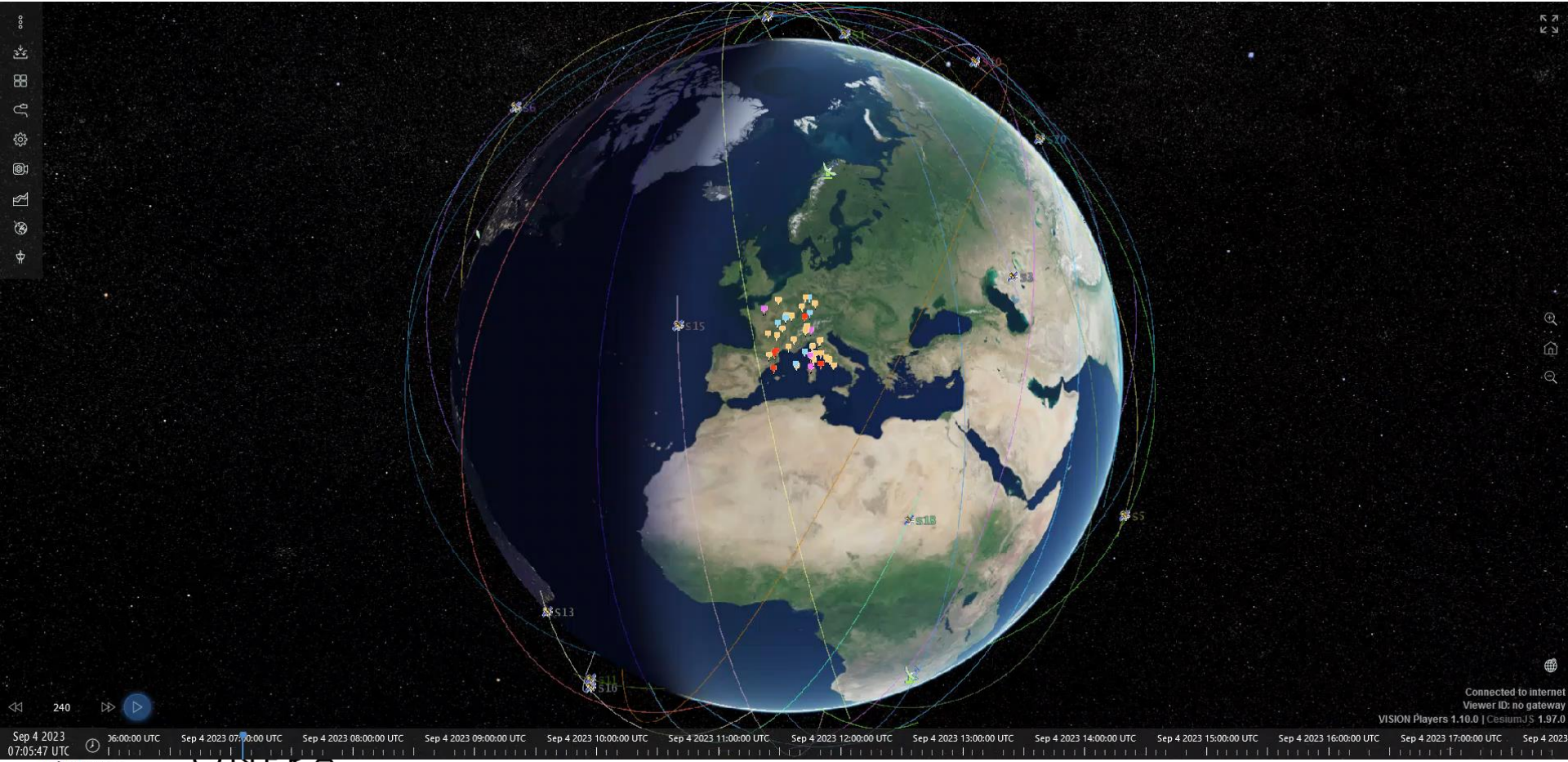
- Satisfaction of a task cluster iff all intervals associated with the required tasks are present
- Formalization : $\forall c \in Clusters,\ satisfied[c] = \min\{\ presence(itv[i]) \mid i \in I(c)\}$

New objective : $maximize\ sum(c \in Clusters)\ satisfied[c] * Reward[c]$

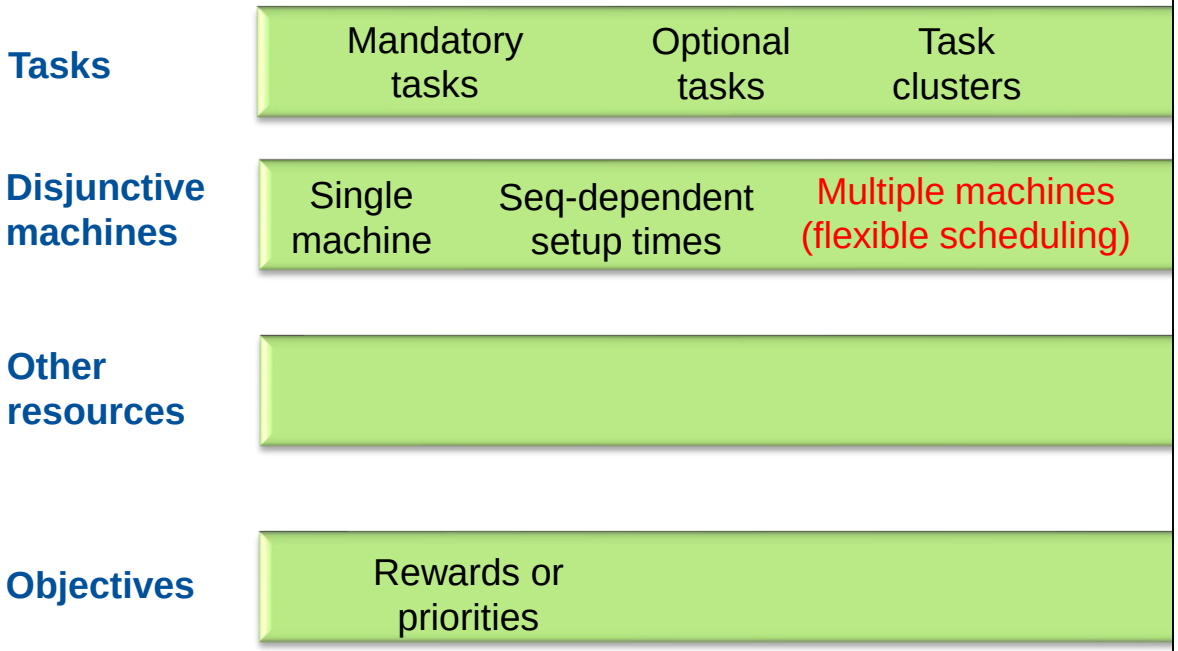
Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times
5. Scheduling with CP : task clusters
6. Scheduling with CP : flexible disjunctive resources

Demonstration (ONERA SpaceLab tool)



Scheduling problems and CP



Problem

- several satellites
- several orbits
- set of possible observation tasks numbered from 1 to N
- for each observation $i \in [1..N]$, associated satellite $Satellite[i]$
- set of obs requests R, with for each $r \in R$ a set of candidate observations $I(r) \subseteq [1..N]$
- time windows
- fixed transition times
- reward Reward[r] for request r

Planning with multiple satellites

Main ideas

- New variables $satisfied[r] \in \{0,1\}$: request r satisfied
- Request satisfaction constraint : $\forall\, r \in R, satisfied[r] = sum\big(i \in I(r)\big)\, presence(itv[i])$
- Definition of one noOverlap constraint per satellite
- Possible decomposition into a larger number of noOverlaps by exploiting long time frames free of candidate obs tasks

```
IloIntVar[] satisfiedRequest = cp.boolVarArray(nRequests);

for(int i=0;i<nRequests;i++) {
    MultiSatRequest r = pb.requests.get(i);
    int nTasks = r.candidateTasks.size();
    IloConstraint[] obsPresences = new IloConstraint[nTasks];

    for(int k=0;k<nTasks;k++) {
        SatelliteTask task = r.candidateTasks.get(k);
        IloIntervalVar itv = cp.intervalVar(task.duration);
        itv.setStartMin(task.earliestStart);
        itv.setStartMax(task.latestStart);
        itv.setOptional();
        obsPresences[k] = cp.presenceOf(itv);
        obsIntervalsForSatellite.get(task.satelliteId).add(itv);
        obsIntervalsForTask.put(task, itv);
    }
    cp.addEq(satisfiedRequest[i], cp.sum(obsPresences));
}

for(int i=0;i<nSatellites;i++)
    cp.add(cp.noOverlap(obsIntervalsForSatellite.get(i)));
```

Planning with multiple satellites

100 tasks, 4 satellites, rewards in set {1,2,3}

```
+ New bound is 189
! Using temporal relaxation.
                    0        402    1           -
+ New bound is 164
*           135      201  0,56s         1      (gap is 21,48%)
            135    1 093          6     1   F          presenceOf(_itv230)
*           139    1 750  0,56s         1      (gap is 17,99%)
            139    2 000          4     1              presenceOf(_itv103)
            139    3 000         15     1   F          presenceOf(_itv241)
            139    1 043          6     2   F          presenceOf(_itv159)
            139    2 000          4     2             !presenceOf(_itv277)
            139    3 000         15     2             !presenceOf(_itv25)
            139    1 057         20     3   F          presenceOf(_itv129)
            139    2 000          1     3             !presenceOf(_itv47)
            139    3 000          7     3       986 != startOf(_itv213)
            139    1 041         10     4   F    76  = startOf(_itv219)
            139    2 000          1     4              presenceOf(_itv137)
            139    3 000          1     4       986 != startOf(_itv213)
            139    1 065         20     5   F          presenceOf(_itv38)
            139    2 000          1     5              presenceOf(_itv78)
            139    3 000          1     5       983  = startOf(_itv151)
            139    1 039         15     6   F          presenceOf(_itv288)
! Time = 0,56s, Average fail depth = 15, Memory usage = 18,4 MB
! Current bound is 164 (gap is 17,99%)
!          Best Branches  Non-fixed    W       Branch decision
            139    2 000          1     6             !presenceOf(_itv144)
            139    3 000          6     6   F   935  = startOf(_itv70)
            139    1 055         20     7   F          presenceOf(_itv110)
            139    2 000          1     7             !presenceOf(_itv299)
            139    3 000         13     7       775  = startOf(_itv67)
            139    1 063         21     8   F          presenceOf(_itv183)
            139    2 000          1     8             !presenceOf(_itv262)
            139    3 000          6     8     1 037 != startOf(_itv81)
            139    1 061          6     9   F   868  = startOf(_itv157)
```

100 tasks, 4 satellites, rewards in set {1,100,10000}

Remarks

- Larger problem size
- Rewards / priorities : strong impact on CPU times (not just a matter of number of observations)
- Window distribution important too

Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times
5. Scheduling with CP : task clusters
6. Scheduling with CP : flexible disjunctive resources
7. Scheduling with CP : reservoir resources

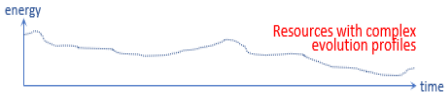
Scheduling problems and CP

Tasks	Mandatory tasks	Optional tasks	Task clusters
Disjunctive machines	Single machine	Seq-dependent setup times	Multiple machines (flexible scheduling)
Other resources	Knapsack-like capacity	Reservoir resource	
Objectives	Rewards or priorities		

Resources !

- min energy level Emin
- max energy level Emax (saturation at Emax)

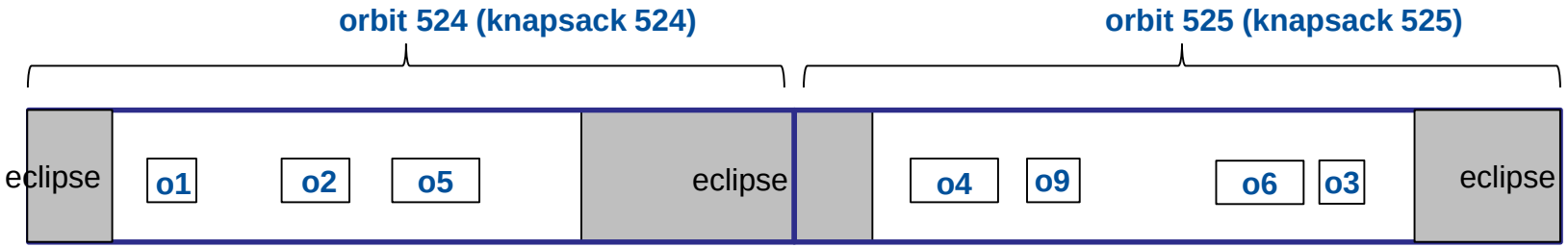
```
▶ tasks:            (30)[ {…},
  start:            0
  timeStep:         10
  nTimeSteps:       300
  spanelPower:      2
  acqPower:         20
  initEnergy:       500
  energyCapacity:   1000
```



Energy management : coarse-grain model

Model 1 : multi-knapsack-like model

- maximum observation time per orbit ($MaxObsDurationPerOrbit$)
- very easy to model in CP given a set of orbits $Orbits$ with for each $o \in Orbits$ the set of indices $I(o)$ of observations involved in o
- capacity constraints :
  $\forall o \in Orbits, sum\big(i \in I(o)\big)\ presence(itv[i]) * DU[i] \leq MaxObsDurationPerOrbit$



Energy management : finer model in CP

Model 2 : reservoir constraint

- Set of resource consumption-production events $\{ (T_k, \delta_k) \mid k \in [1..M] \}$
  regular energy production by the solar panels
  + energy reservation for obs and downloads
- Minimum energy level Emin
- Maximum energy Emax

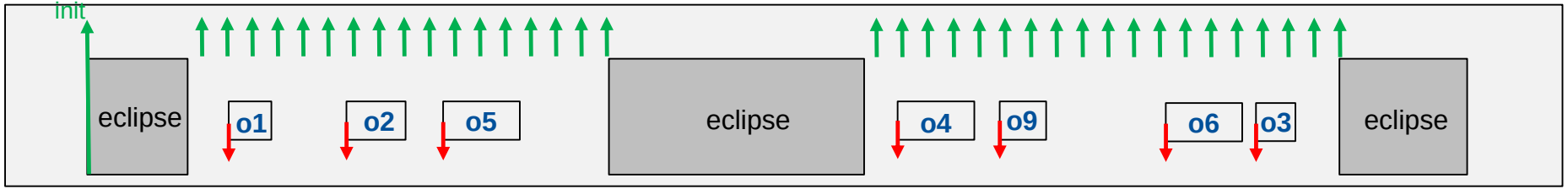
Does not model the saturation at Emax
Ex : possible inconsistency if only charging activities leading to production beyond Emax !!!

Semantics : at every time t,
$\text{Emin} \leq sum(k \in [1..M]\ s.t. T_k \leq t)\ \delta_k \leq Emax$

Model 3: reservoir constraint with optional events

- Set of optional resource prod/cons events $\{ (T_k, \delta_k, x_k) \mid k \in [1..M] \}$ where $x_k \in \{0,1\}$ = presence of event k
  power production only if needed
- Minimum energy level Emin
- Maximum energy Emax

Semantics : at every time t,
$\text{Emin} \leq sum(k \in [1..M]\ s.t. T_k \leq t)\ x_k * \delta_k \leq Emax$



Energy management

CPOptimizer (Java API)

```
IloCP cp = new IloCP();
int nTasks = pb.tasks.size();
IloIntervalVar[] obsItvs = new IloIntervalVar[nTasks];
for(int i=0;i<nTasks;i++) {
    Task task = pb.tasks.get(i);
    IloIntervalVar itv = cp.intervalVar(task.duration);
    itv.setStartMin(task.earliestStart);
    itv.setStartMax(task.latestStart);
    itv.setOptional();
    obsItvs[i] = itv;
}
cp.add(cp.noOverlap(obsItvs));

IloCumulFunctionExpr reservoir = cp.cumulFunctionExpr();
reservoir.add(cp.step(pb.start, pb.initEnergy));
for(int i=0;i<nTasks;i++) {
    reservoir.sub(cp.stepAtStart(obsItvs[i],
            pb.acqPower * pb.tasks.get(i).duration));
}
for(int i=0;i<pb.nTimeSteps;i++) {
    IloIntervalVar itv = cp.intervalVar(pb.timeStep);
    itv.setStartMin(pb.start + i*pb.timeStep);
    itv.setStartMax(pb.start + i*pb.timeStep);
    itv.setOptional();
    reservoir.add(cp.stepAtEnd(itv, pb.spanelPower * pb.timeStep));
}
cp.add(cp.alwaysIn(reservoir, pb.start, pb.end, 0, pb.energyCapacity));
```

initial energy

energy consumption

energy production

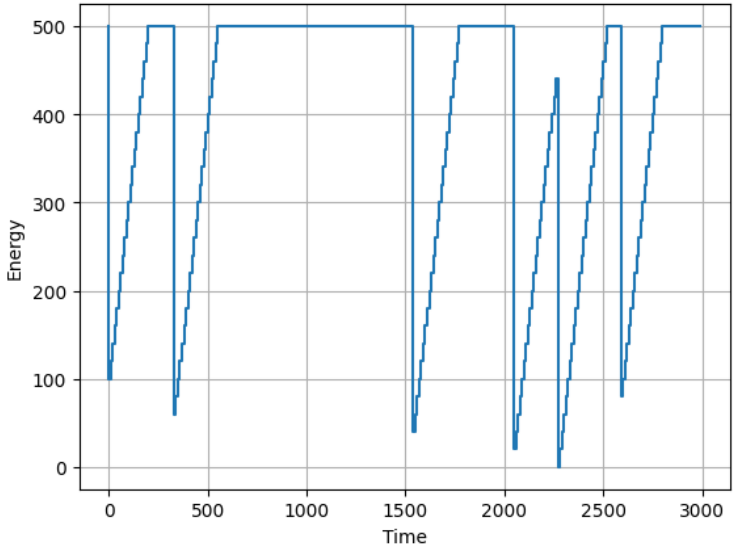
reservoir constraint

OR-Tools CP-SAT (Python API)

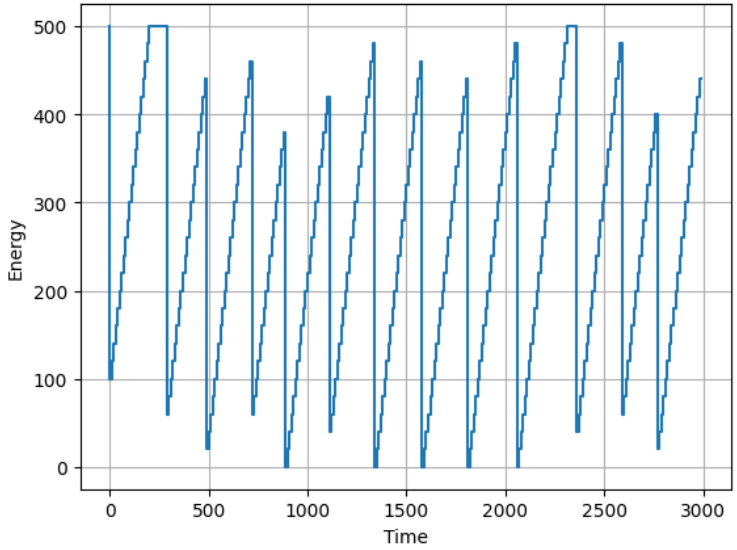
```
times.append(startTime)
demands.append(initEnergy)
actives.append(model.NewConstant(1))

for task in tasks:
    (name,est,lst,du) = (task['name'], task['earliestStart'],
                         task['latestStart'], task['duration'])
    start = model.NewIntVar(est, lst, f"start({name})")
    end = model.NewIntVar(est + du, lst + du, f"end({name})")
    presence = model.NewBoolVar(f"presence({name})")
    interval = model.NewOptionalIntervalVar(start, du, end, presence, f"itv({name})")
    presences.append(presence)
    intervals.append(interval)

    times.append(start)
    demands.append(- acqPower * du)
    actives.append(presence)

for k in range(nTimeSteps):
    times.append(startTime + k * timeStep)
    demands.append(spanelPower * timeStep)
    actives.append(model.NewBoolVar(f"charge({k})"))

model.AddNoOverlap(intervals)
model.AddReservoirConstraintWithActive(times, demands, actives, 0, energyCapacity)
```

Energy management : coarse-grain model

Solution example (using CPOptimizer),
20 candidate observations



Solution example (using CPOptimizer),
150 candidate observations



Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times
5. Scheduling with CP : task clusters
6. Scheduling with CP : flexible disjunctive resources
7. Scheduling with CP : reservoir resources
8. Scheduling with CP : cumulative resources

Scheduling problems and CP

Tasks	Mandatory tasks	Optional tasks	Task clusters	Tasks with precedences
Disjunctive machines	Single machine	Seq-dependent setup times	Multiple machines (flexible scheduling)	
Other resources	Knapsack-like capacity	Reservoir resource	Cumulative resource	
Objectives	Rewards or priorities			

Resources again !

- Maximum memory size
- Memory reserved at the start of an observation
- Memory released at the end of each download task

Memory management : possible model

Cumulative constraint with optional intervals

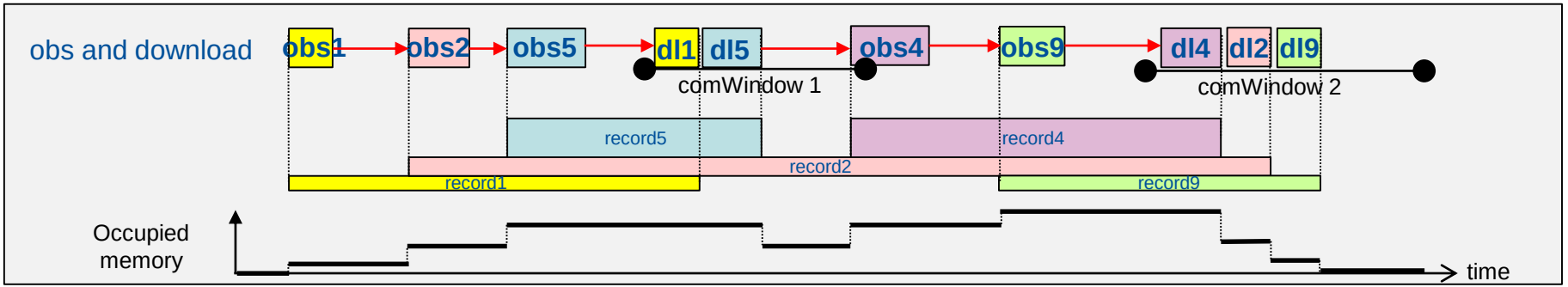
- set of tasks K
- optional intervals $itv[k]$ for each task $k \in K$
- resource consumption $Rcons[k]$ for each k
- maximum resource level $Rmax$

Semantics :

$$\forall t, \sum_{\substack{k \in K \; s.t. \; presence(itv[k])=1 \\ AND \; start(itv[k]) \leq t < end(itv[k])}} Rcons[k] \leq Rmax$$

Application

- optional intervals $recordItv[i]$ for each obs $i \in [1..N]$
- $startAtStart(recordItv[i], obsItv[i])$
- $endAtEnd(recordItv[i], dlItv[i])$
- $endBeforeStart(obsItv[i], dlItv[i])$
- Cumulative constraint over intervals $recordItv[i]$ given maximum memory size $MemMax$
- + a few other constraints (alternative, span…)



Energy management

CPOptimizer (Java API)

```
IloCumulFunctionExpr occupiedMemory = cp.cumulFunctionExpr();
for(int i=0;i<nTasks;i++) {
    occupiedMemory = cp.sum(occupiedMemory,
            cp.pulse(recordItvs[i], pb.obsTasks.get(i).memSize));
}
cp.add(cp.le(occupiedMemory, pb.memCapacity));
```

OR-Tools CP-SAT (Python API)

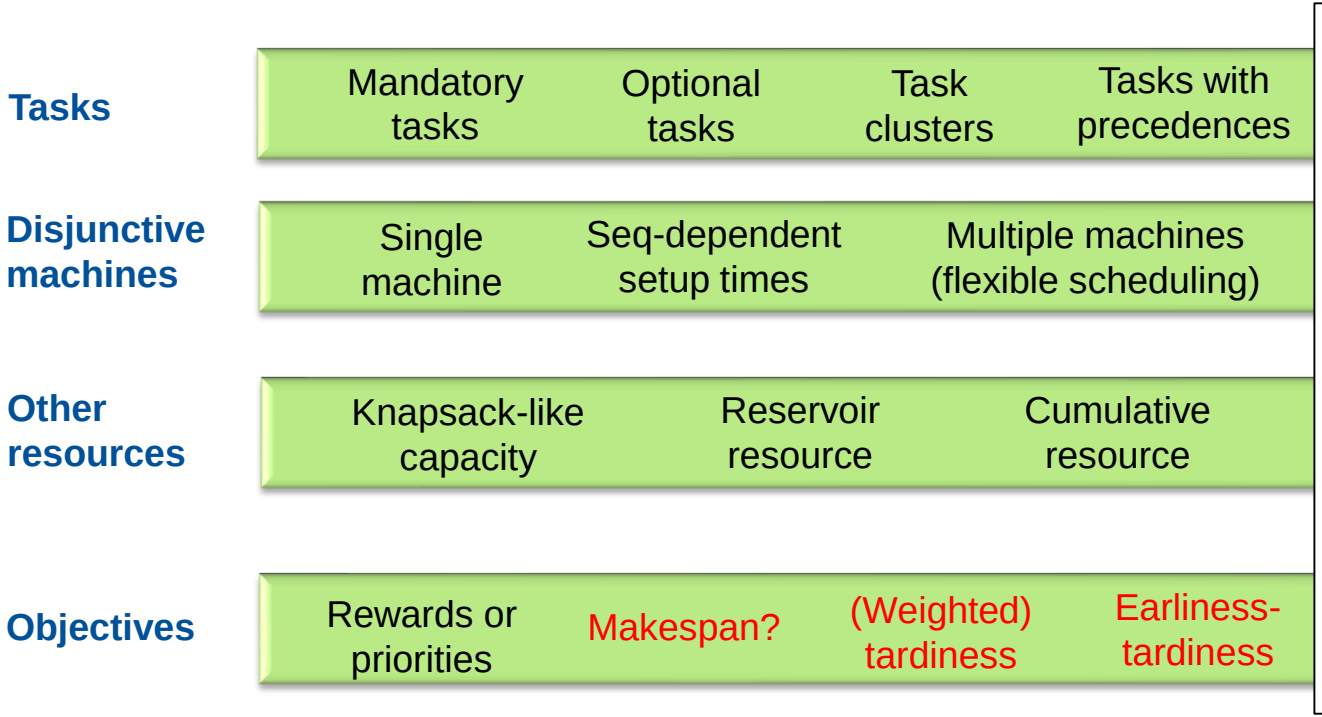
```
model.AddCumulative(intervals, demands, capacity)
```

Powerful propagation algorithms associated with the cumulative constraint

Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times
5. Scheduling with CP : task clusters
6. Scheduling with CP : flexible disjunctive resources
7. Scheduling with CP : reservoir resources
8. Scheduling with CP : cumulative resources
9. Scheduling with CP : other objectives

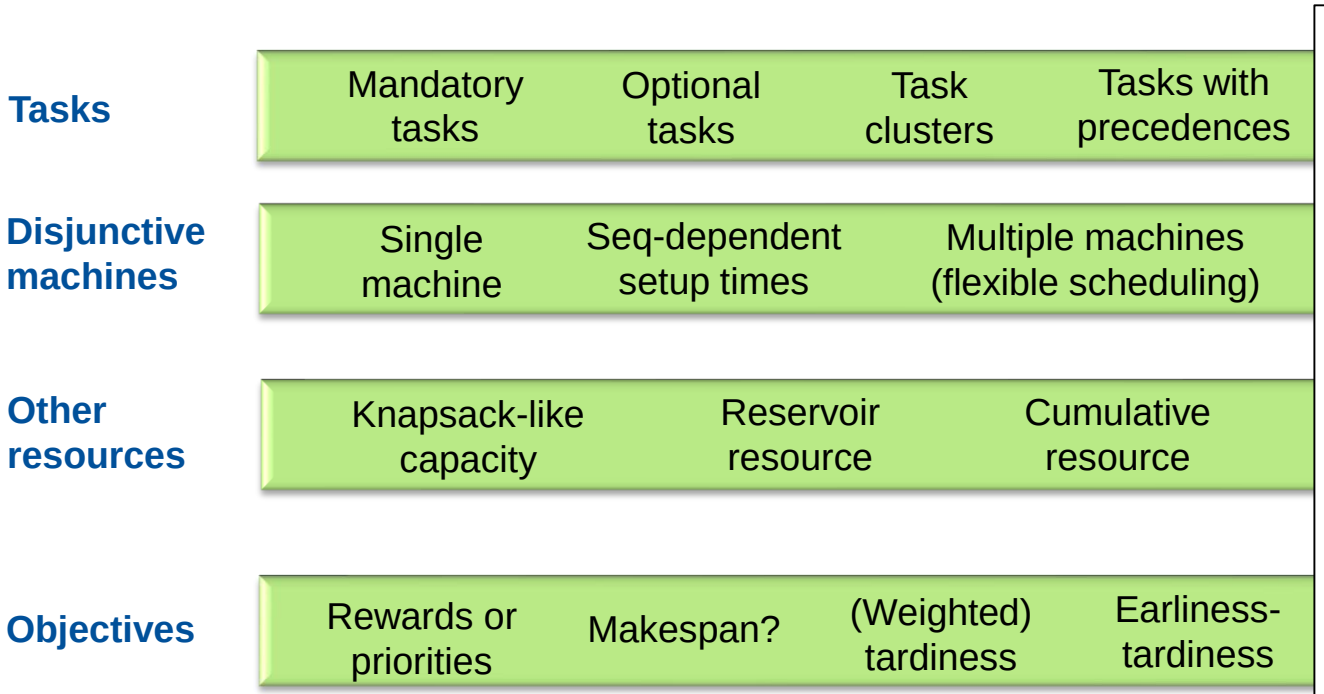
Scheduling problems and CP



Other objective functions ?

Rather easy to model in CP

Scheduling problems and CP



Strength of Constraint Programming for modeling and solving many scheduling problems !

Everything is green…

However…

Computational aspects

Possible scalability issues

- with increases in the problem size
- for specific input data (see the previous examples concerning the reward distribution)

Operational requirements (also true for other application fields)

- need for fast response times, typically a few minutes
- need for robustness (not acceptable to have no solution at all)

Possible strategies

- development of new efficient propagation algorithms to better prune the search space
- new search methods combining optimization and learning (clause learning, parameter learning, etc.)
- CP and (meta)heuristics run in parallel to get at least a solution
- Hybrid methods such as Large Neighborhood Search (LNS)

Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times
5. Scheduling with CP : task clusters
6. Scheduling with CP : flexible disjunctive resources
7. Scheduling with CP : reservoir resources
8. Scheduling with CP : cumulative resources
9. Scheduling with CP : other objectives
10. CP-based Large Neighborhood Search

Large neighborhood search

LNS in general

- Start from a full solution (possibly obtained by greedy search)
- Destroy a part of the solution and repair the new solution obtained
- Iterate this process until a termination condition (e.g., maximum nb of iterations without improvement, maximum CPU time)

CP-based LNS

- Start from a full assigment A of the decision variables in X (the problem variables)
- Destroy : unassign k variables $\{y_1, \dots, y_k\}$ in X
- Repair : use a CP solver to re-optimize the values of $y_1, \dots, y_k$ given the fixed values of the other variables
- Iterate this process until a termination condition (e.g., maximum nb of iterations without improvement, maximum CPU time)

CP-based LNS

Several destroy strategies for scheduling problems

- Random removal of k tasks
- Destroy activities scheduled over a specific time interval
- Destroy all activities scheduled over some machines
- Destroy activities scheduled over a specific time interval over a subset of the machines
- Destroy some precedence constraints between the activities (to get a partial ordering)
- …

Examples of approaches combining CP and LNS / LocalSearch for EOS planning

- S. Squillaci, C. Pralet, and S. Roussel. 2023. Scheduling Complex Observation Requests for a Constellation of Satellites: Large Neighborhood Search Approaches. In CPAIOR 2023.
- V. Antuori, D. T. Wojtowicz, E. Hebrard. Solving the Agile Earth Observation Satellite Scheduling Problem with CP and Local Search. In CP 2025.

CP-based LNS : several approaches

Approach 1

- Define a global CP model over X
- At each step, assign all variables in $X \setminus \{y_1, \dots, y_k\}$ and solve the global model
- Advantage : reuses the basic CP model of the problem, almost no additional implementation effort

Approach 2

- Define a local CP model encoding the neighborhood more compactly, possibly using a small problem holding only over variables in $\{y_1, \dots, y_k\}$
- At each step, tune the local CP model, solve it, and merge the local solution with the global one
- Advantage : smaller problem (lower memory size)

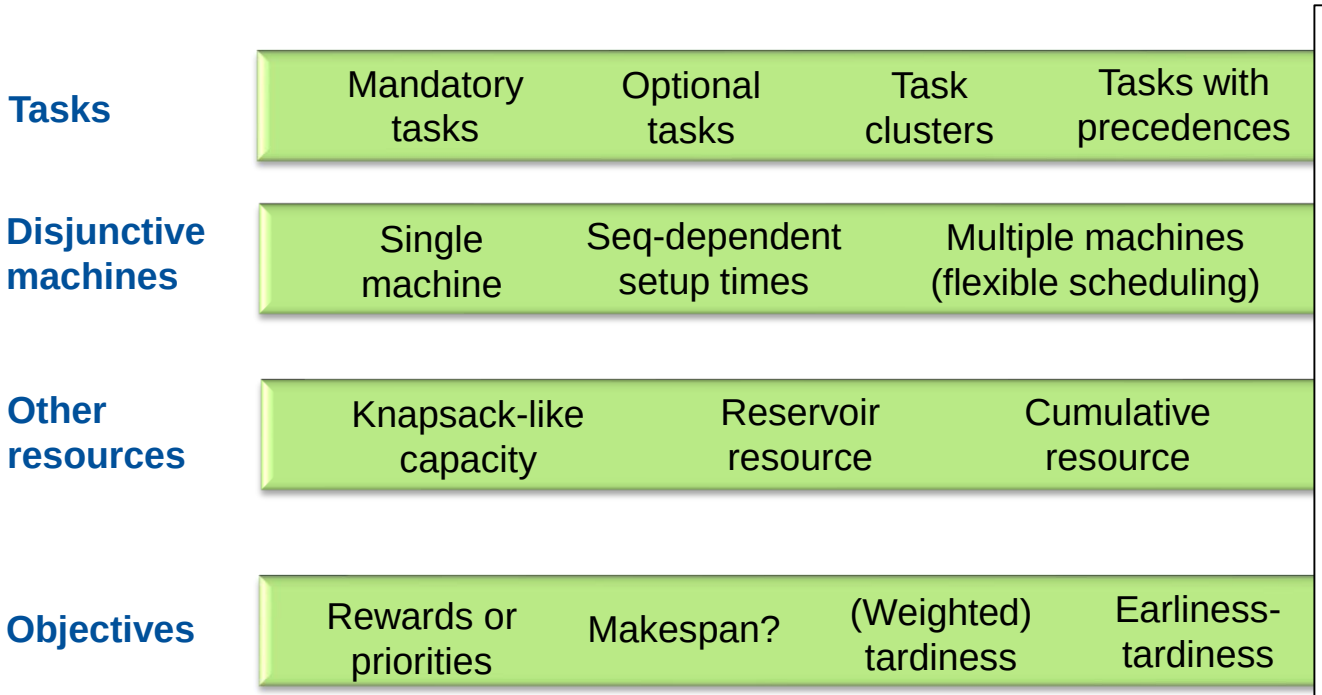
Approach 3

- Use a CP solver that already disposes of LNS search strategies (e.g., CPOptimizer or OR-Tools CP-SAT)

Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times
5. Scheduling with CP : task clusters
6. Scheduling with CP : flexible disjunctive resources
7. Scheduling with CP : reservoir resources
8. Scheduling with CP : cumulative resources
9. Scheduling with CP : other objectives
10. CP-based Large Neighborhood Search
11. Modeling issues

Scheduling problems and CP



Strength of Constraint Programming for modeling and solving many scheduling problems !

Techniques like LNS can help for scalability issues

Everything is green again…

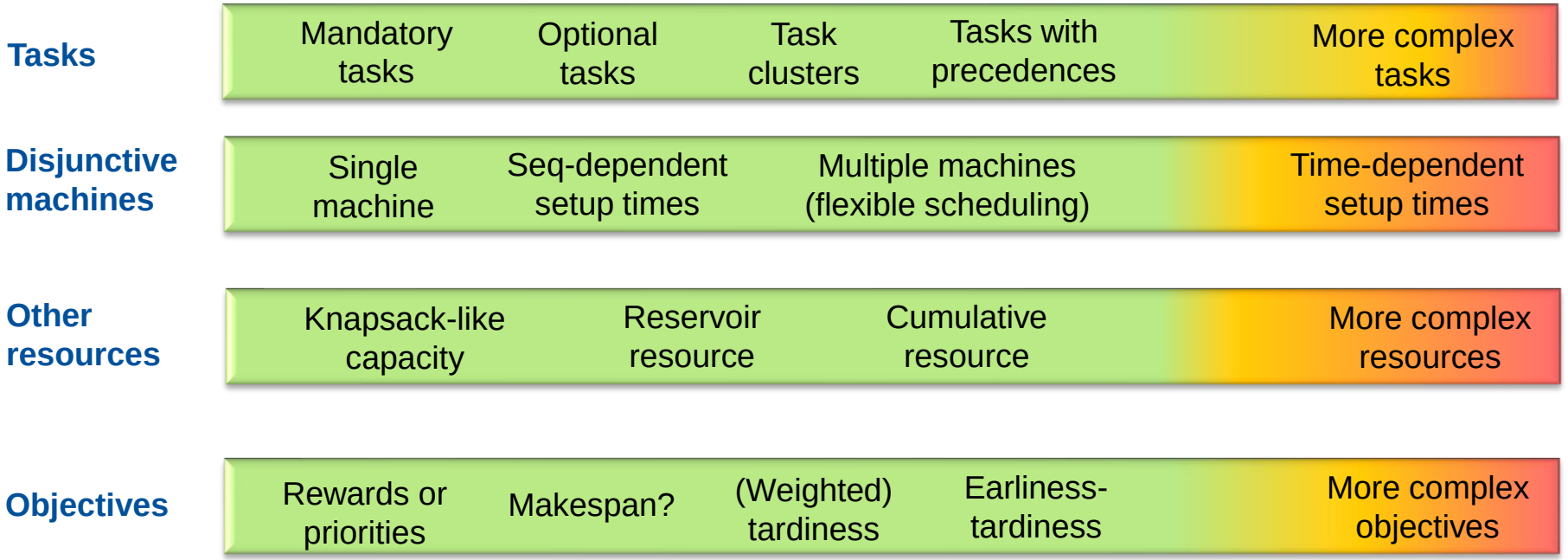
However…

Scheduling problems and CP

Tasks	Ex : periodic observation tasks, coverage tasks for large areas…	More complex tasks
Disjunctive machines	Need to consider kinematical models (max velocity / acceleration of the gyroscopic actuators) coupled with space mechatronics	Time-dependent setup times
Other resources	Ex : recharge function of the cosine of the angle between the Sun rays and the normal to the solar panels	More complex resources
Objectives	Ex : combination of repetitivity objectives and cloud cover aspects	More complex objectives

Operational baseline for EOS planning : greedy hierarchical search

Scheduling problems and CP



Organization

1. Constraint Programming (CP) : generalities
2. Scheduling with CP : noOverlap
3. Scheduling with CP : noOverlap with optional tasks
4. Scheduling with CP : noOverlap with optional tasks and setup times
5. Scheduling with CP : task clusters
6. Scheduling with CP : flexible disjunctive resources
7. Scheduling with CP : reservoir resources
8. Scheduling with CP : cumulative resources
9. Scheduling with CP : other objectives
10. CP-based Large Neighborhood Search
11. Modeling issues
12. Conclusion and perspectives

Conclusion and perspectives

Good news

- CP very flexible to model problems, thanks to expressive temporal and resource constraints over time or interval variables, coupled with Boolean conditions
- Very efficient propagation techniques developed during decades (noOverlap constraints, cumulative constraints, efficient CP-SAT-temporal network reasoning, lazy clause generation…)
- CP efficient on small-medium-size simplified EOS scheduling problems
- Still work to be done given remaining scalability and modeling issues discussed before
- And anyway, CP is a very good candidate to get optimal solution to small-medium simplified problems and evaluate the optimality gaps of (meta)heuristic methods
- Other relevant space applications (e.g., telecommunication satellites, telescopes, rovers, deep space exploration…)

Conclusion and perspectives

Some perspectives

- hybrid methods (clause learning coupled with detailed models, optimization and machine learning, new CP-based LNS strategies, hybrization between CP and other metaheuristics…)
- development of new specific constraints closer to some complex physical models
- Uncertainty management
- Onboard EOS planning
- …

Several ongoing projects, such as ANR OMNEOS involving ONERA, LAAS-CNRS, and Prométhée

Some references…

- Antuori, V., Wojtowicz, D. T., Hebrard, E, Solving the Agile Earth Observation Satellite Scheduling Problem with CP and Local Search. In CP 2025.
- Barrault, R., Pralet, C., Picard, G., Sawyer, G.. 2025. Hybridizing Machine Learning and Optimization for Planning Satellite Observations. In Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 22nd International Conference, CPAIOR 2025.
- Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using Iterative Repair to Improve Responsiveness of Planning and Scheduling. In International Conference on Artificial Intelligence Planning Systems (AIPS 2000).
- Du, Y., Wang, T., Xin, B., Wang, L., Chen, Y., Xing, L.: A data-driven parallel scheduling approach for multiple agile Earth observation satellites. IEEE Transactions on Evolutionary Computation 24(4), 679–693 (2020)
- Eddy, D., Kochenderfer, M.: A maximum independent set method for scheduling Earth-observing satellite constellations. Journal of Spacecraft and Rockets 58, 1–14 (2021)
- Fruth, T.; Lenzen, C.; Gross, E.; and Mrowka, F. 2018. The EnMAP Mission Planning System. In International Conference on Space Operations (SpaceOps).
- He, L., de Weerdt, M., Yorke-Smith, N.: Time/sequence-dependent scheduling: the design and evaluation of a general purpose tabu-based adaptive large neighbourhood search algorithm. Journal of Intelligent Manufacturing 31(4), 1051–1078 (2020)
- He, L., de Weerdt, M., Yorke-Smith, N., Liu, X., Chen, Y.: Tabu-based large neighbourhood search for time-dependent multi-orbit agile satellite scheduling. In: Proc. of the 11th Scheduling and Planning Applications Workshop (SPARK) (2018)
- Hu, X., Zhu, W., An, B., Jin, P., Xia, W.: A branch and price algorithm for EOS constellation imaging and downloading integrated scheduling problem. Computers & Operations Research 104, 74–89 (2019)
- Kim, J., Ahn, J., Choi, H.L., Cho, D.H.: Task scheduling of multiple agile satellites with transition time and stereo imaging constraints. Journal of AerospaceInformation Systems 17(6) (2020)
- Laborie, P.: IBM ILOG CP optimizer for detailed scheduling illustrated on three problems. In: van Hoeve, W.J., Hooker, J.N. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, vol. 5547, pp. 234–235 (2009)
- Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG 23(2), 210–250 (2018)
- Liu, L., Dong, Z., Su, H., Yu, D., Lin, Y.: Research on a heterogeneous multi-satellite mission scheduling model for Earth observation based on adaptive genetic-tabu hybrid search algorithm. In: IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC). pp. 1684–1690 (2021)
- Liu, X., Laporte, G., Chen, Y., He, R.: An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. Computers & Operations Research 86, 41–53 (2017)
- Parjan, S.; and Chien, S. 2023. Decentralized Observation Allocation for a Large-Scale Constellation. Journal of Aerospace Information Systems (JAIS), 20(8): 447–461.

Some references…

- Peng, G., Dewil, R., Verbeeck, C., Gunawan, A., Xing, L., Vansteenwegen, P.: Agile Earth observation satellite scheduling: An orienteering problem with time- dependent profits and travel times. Computers & Operations Research 111, 84–98 (2019)
- Picard, G.: Auction-based and distributed optimization approaches for scheduling observations in satellite constellations with exclusive orbit portions. International Conference on Autonomous Agents and Multiagent Systems (2021)
- Pralet, C.: Iterated maximum large neighborhood search for the traveling salesman problem with time windows and its time-dependent version. Computers &Operations Research 150 (2022)
- Pralet, C., Doose, D., Anxionnat, J., & Pouly, J. (2022). Building Resource-Dependent Conditional Plans for an Earth Monitoring Satellite. *Proceedings of the International Conference on Automated Planning and Scheduling*, *32*(1), 490-498. https://doi.org/10.1609/icaps.v32i1.19835
- Pralet, C.: Continuous Planning and Execution for a Mission Planning System Managing a Constellation of Earth Observation Satellites. International Workshop on Planning and Scheduling for Space (IWPSS), (2025)
- Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; and Govindjee, A. 1999. Iterative Repair Planning for Spacecraft Operations in the ASPEN System. In International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS 1999).
- Schmid, V., Ehmke, J.F.: An effective large neighborhood search for the team orienteering problem with time windows. In: International Conference on Computational Logistics. pp. 3–18. Springer (2017)
- Squillaci, S.; Pralet, C.; and Roussel, S. 2023. Scheduling Complex Observation Requests for a Constellation of Satellites: Large Neighborhood Search Approaches. In 20th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 443–459.
- Squillaci, S., Pralet, C., Roussel, S. Comparison of time-dependent and time-independent scheduling approaches for a constellation of Earth observing satellites. In IWPSS 2023
- Valicka, C.; Garcia, D.; Staid, A.; Watson, J.-P.; Hackebeil, G.; Rathinam, S.; and Ntaimo, L. 2019. Mixed-Integer Programming Models for Optimal Constellation Scheduling Given Cloud Cover Uncertainty. European Journal of Operational Research, 275(2): 431–445.
- Wang, J.; Demeulemeester, E.; Hu, X.; Qiu, D.; and Liu, J. 2019. Exact and Heuristic Scheduling Algorithms for Multiple Earth Observation Satellites Under Uncertainties of Clouds. IEEE Systems Journal, 13(3).
- Wei, L., Xing, L., Wan, Q., Song, Y., Chen, Y.: A multi-objective memetic approach for time-dependent agile Earth observation satellite scheduling problem. Computers & Industrial Engineering 159 (2021)
- Wu, G., Wang, H., Pedrycz, W., Li, H., Wang, L.: Satellite observation scheduling with a novel adaptive simulated annealing algorithm and a dynamic task clustering strategy. Computers & Industrial Engineering 113, 576–588 (2017)

Questions ?