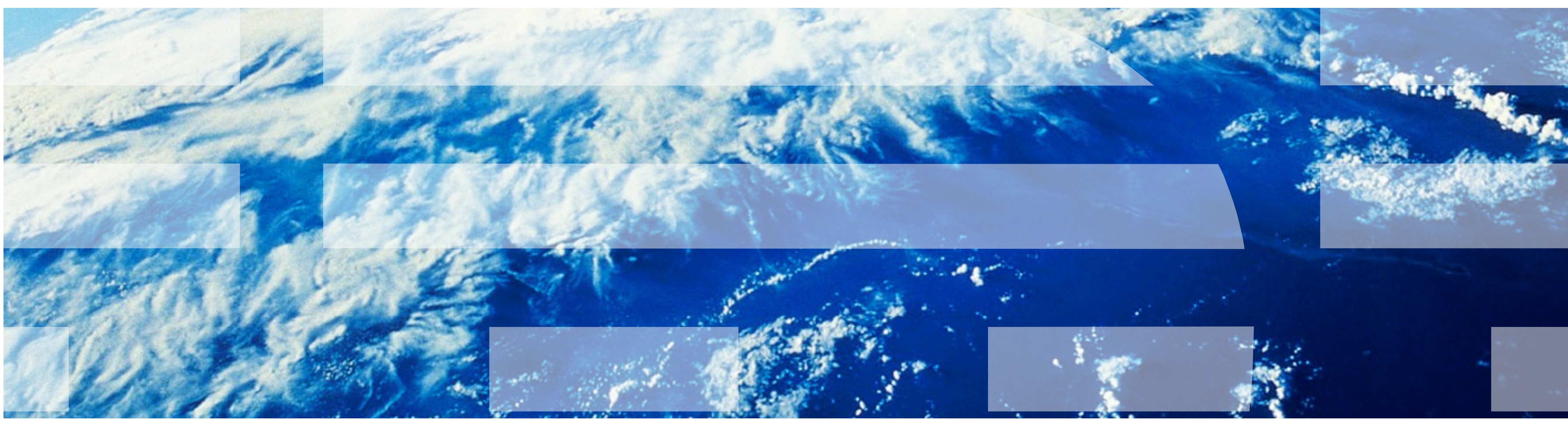
Modeling and Solving Scheduling Problems with CP Optimizer

Philippe Laborie
CPLEX Optimization Studio Team



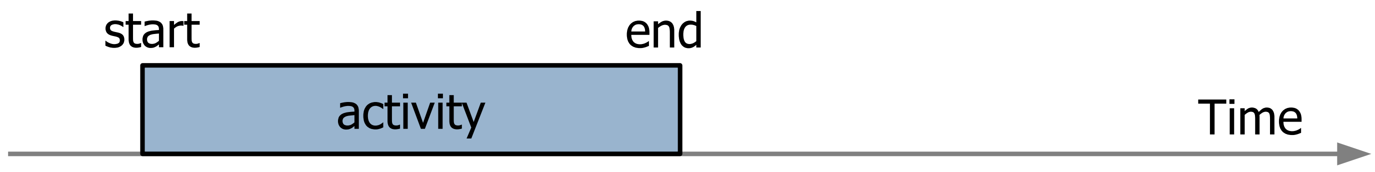
May 28, 2014

Agenda

- IBM ILOG **CP Optimizer** for **Scheduling**
- Scheduling **concepts** in CP Optimizer
- **Tips** and **Tools**
- **Q&A**

IBM ILOG CP Optimizer for Scheduling

- Scheduling consist of assigning **starting** and **completion times** to a set of activities while satisfying different types of constraints (resource availability, precedence relationships, …) and optimizing some criteria (minimizing tardiness, …)



- Time is considered as a continuous dimension: domain of possible start/completion times for an activity is potentially **very large**
- Beside start and completion times of activities, other types of decision variables are often involved in real industrial scheduling problems (resource **allocation**, **optimal** activities …)

IBM ILOG CP Optimizer for Scheduling

- Modeling and solving scheduling problems

Model & run paradigm	User-implemented resolution
MIP (CPLEX)	dispatching rules, heuristics, meta-heuristics

IBM ILOG CP Optimizer for Scheduling

- Modeling and solving scheduling problems

Model & run paradigm		User-implemented resolution
MIP (CPLEX)	CP Optimizer for Scheduling	dispatching rules, heuristics, meta-heuristics

- Dedicated modeling concepts for scheduling
  - Consistent with Optimization paradigm: new decision variables, expressions and constraints
  - Available in OPL and APIs (C++, Java, .NET)
- Automatic search
  - Complete
  - Combines tree search, large neighborhood search, meta-heuristics, relaxations, learning, ...

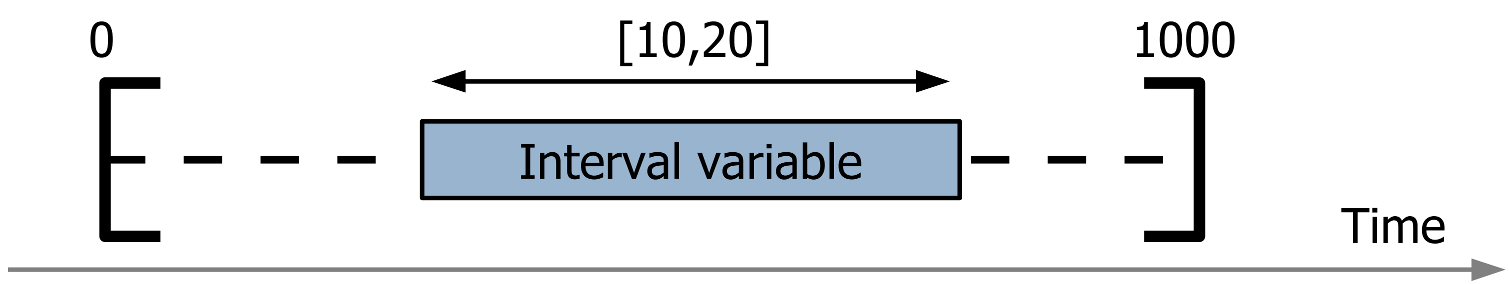
Agenda

- **IBM ILOG CP Optimizer** for **Scheduling**
- Scheduling **concepts** in CP Optimizer
- **Tips** and **Tools**
- **Q&A**

Concept: **interval variable**

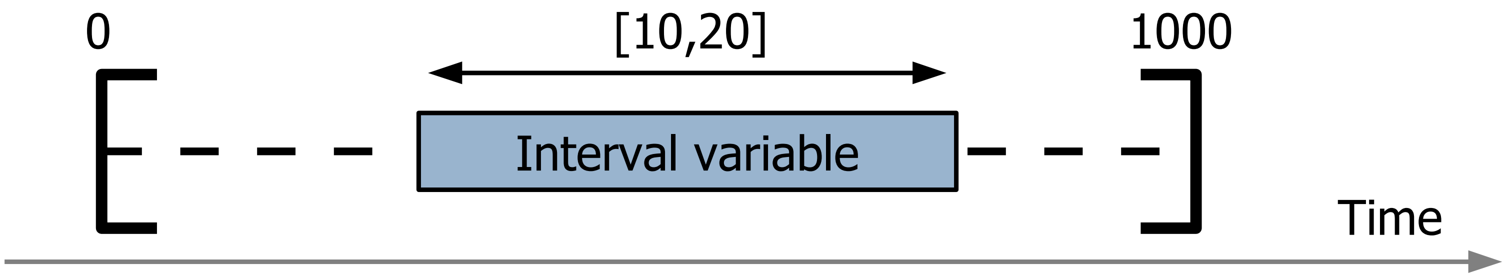
- What for?
  - modeling an interval of time during which a particular property holds (an activity executes, a resource is idle, a tank must be non-empty, …)
- Example:

```
dvar interval x in 0..1000 size in 10..20
```



Concept: **interval variable**

```
dvar interval x in 0..1000 size in 10..20
```



- Properties:
  - The **value** of an interval variable is an integer interval [start,end)
  - **Domain** of possible values: [0,10), [1,11), [2,12),... [990,1000), [0,11),[1,12),...
  - Domain of interval variables is represented **compactly** in CP Optimizer (a few bounds: smin, smax, emin, emax, szmin, szmax)

Concepts: **optional interval variable**

- Interval variables can be defined as being **optional** that is, it is part of the decisions of the problem to decide whether the interval will be **present** or **absent** in the solution
- What for?
  - Modeling optional activities, alternative execution modes for activities, and … most of the discrete decisions in a schedule
- Example:

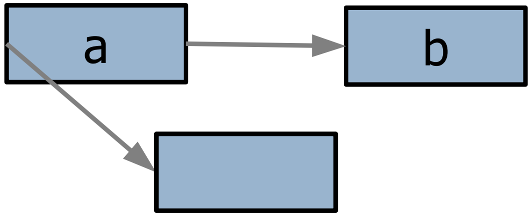
```
dvar interval x optional in 0..1000
                size in 10..20
```

- Properties:
  - An optional interval variable has an additional possible value in its domain (absence value)
  - **Optionality** is a powerful property that you must learn to leverage in your models (more on this later ...)

Concept: **precedence constraint**

- What for ?
  - Modeling temporal constraints between interval variables

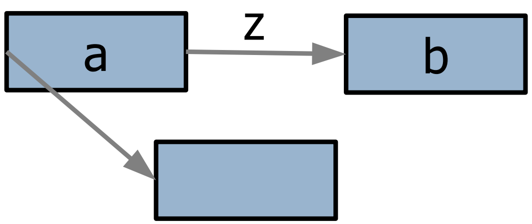
- Example:




```
endBeforeStart(a,b)
startBeforeStart(a,b)
startBeforeEnd(a,b)
endBeforeEnd(a,b)
endAtStart(a,b)
startAtStart(a,b)
startAtEnd(a,b)
endAtEnd(a,b)
```

Concept: **precedence constraint**

- What for ?
  - Modeling temporal constraints between interval variables
  - Modeling constant or variable minimal delays
- Example:



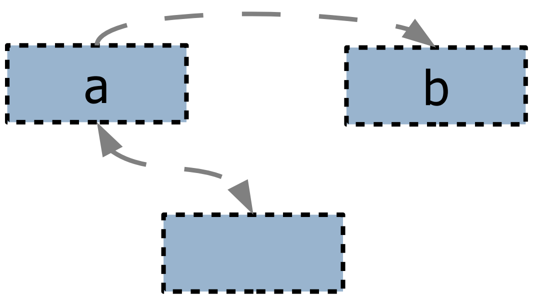

```
endBeforeStart(a,b,z)
startBeforeStart(a,b,z)
startBeforeEnd(a,b,z)
endBeforeEnd(a,b,z)
endAtStart(a,b,z)
startAtStart(a,b,z)
startAtEnd(a,b,z)
endAtEnd(a,b,z)
```

Concept: **precedence constraint**

- Properties
  - Semantic of the constraints handles optionality (as for all constraints in CP Optimizer). Example of endBeforeStart:
    present(a) AND present(b) $\Rightarrow$ end(a)+z $\leq$ start(b)
  - All precedence constraints are aggregated in a temporal network and handled by dedicated graph algorithms (fast global propagation, negative cycle detection, ...)

Concept: **presence constraint**

- What for:
  - Expressing dependency constraints between execution of optional activities or between allocated resources …
- Examples:




```
 presenceOf(a) =>  presenceOf(b)
 presenceOf(a) ==  presenceOf(b)
 presenceOf(a) => !presenceOf(b)
!presenceOf(a) =>  presenceOf(b)
```

Concept: **presence constraint**

- Properties:
  - All binary constraints between presence status are aggregated in an implication graph
  - CP Optimizer maintains hypothetical bounds on interval variables (e.g. start min would the interval be present)
  - CP Optimizer exploits the implication graph to perform conditional reasoning between related interval variables

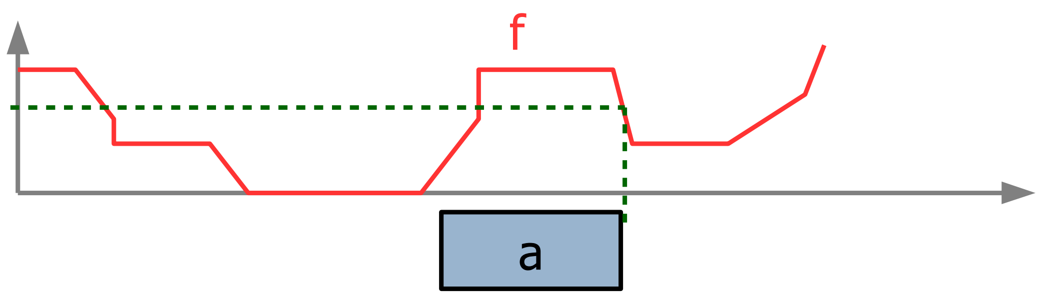
Concept: **expressions on interval variables**

- What for?
  - Evaluating a characteristic of an interval variable (start, end, size) for use in the objective function or in a constraint
- Example:
  - endOf(a, ABSVAL) takes value ABSVAL if interval *a* is absent otherwise it takes value *e* if *a* is present and *a=[s,e)*
  - Typical makespan expression: max(i in 1..n) endOf(a[i])

Concept: **expressions on interval variables**

- What for?
  - Evaluating a piecewise linear function at a particular end-point of an interval variable (earliness-tardiness cost, temporal preference)

- Example:
  - endEval(f,a, ABSVAL) takes value ABSVAL if *a* is absent otherwise it takes value f(*e*) if *a* is present and *a=[s,e)*



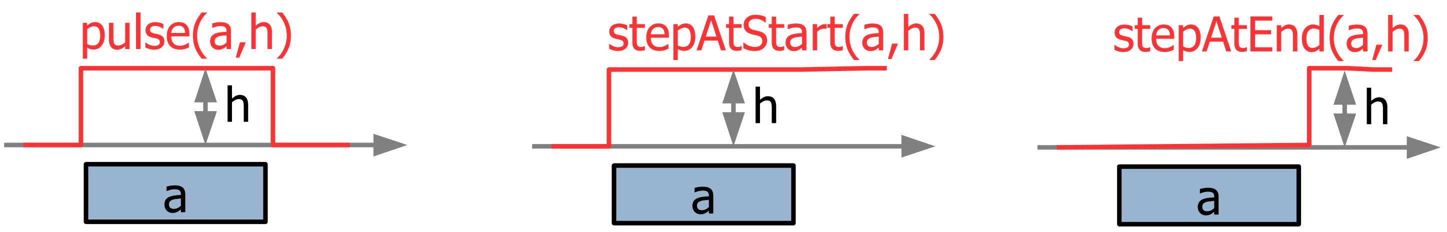
- Property:
  - f can be any piecewise linear function (non-continuous, non-convex, ...)

Concept: **cumul function**

- What for?
  - Modeling and constraining cumulative quantities over time (cumulative use of a discrete resource, level of an inventory with producing and consuming tasks)
  - Restricting the number of intervals that overlap a given date t
  - Forcing a minimal number of intervals to overlap a given date t
  - Complex synchronization constraints between interval variables: e.g. between activities producing/consuming in a tank and the set of time-intervals during which the tank is non-empty

Concept: **cumul function**

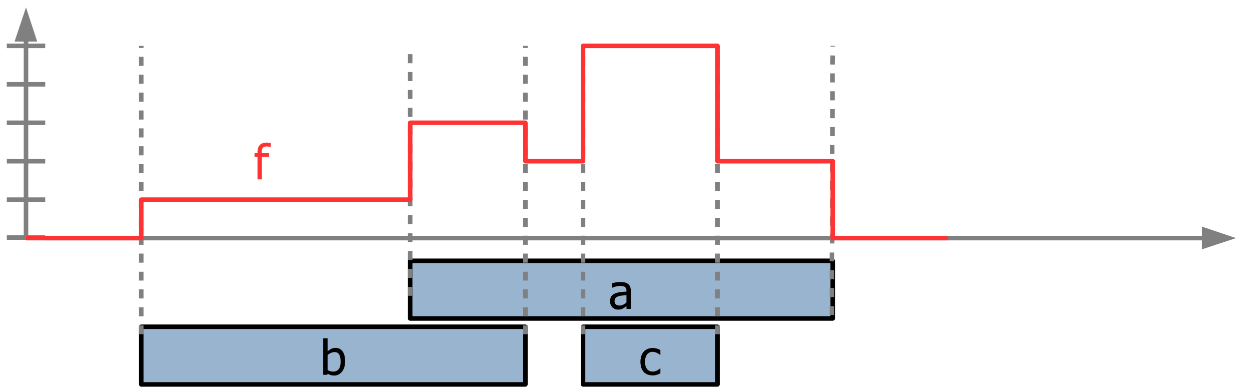
- Cumul functions are built from atomic functions



- A cumul function is the sum of atomic functions or their opposite

Concept: **cumul function**

- Examples:
  – cumulFunction f = pulse(a,2)+pulse(b,1)+pulse(c,3)



  – Constraint f<=C (global capacity limit)
  – Constraint alwaysIn(f,t0,t1,Cmin,Cmax) (capacity profile)
  – Constraint alwaysIn(f,x,Cmin,Cmax) (condition holds over an interval variable x)

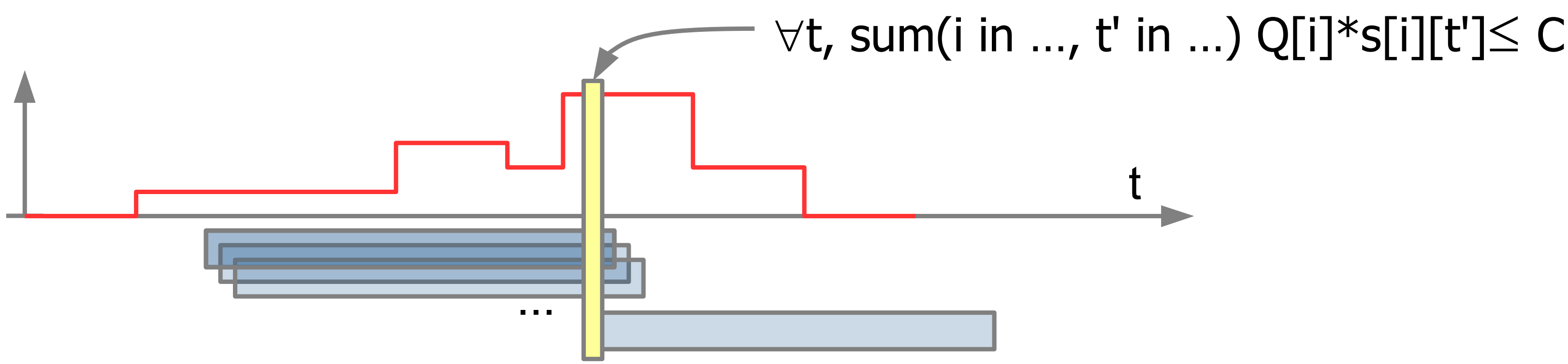
Concept: **cumul function**

- Properties:
  - Complexity of cumul functions is **independent of the time scale**
  - CP Optimizer is able to reason globally over cumul functions

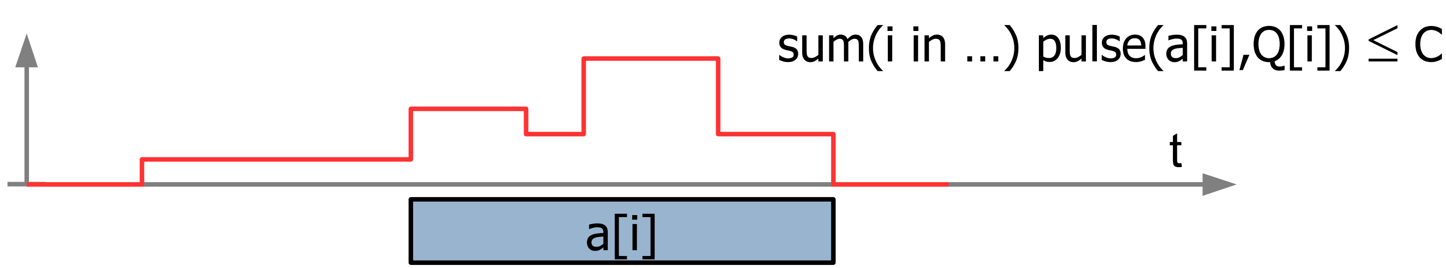
Concept: **cumul function**

- Properties:
  - Compare:
    - Time-indexed MIP model:

`s[i][t]∈{0,1}: s[i][t]=1 iff a[i] starts at date t`

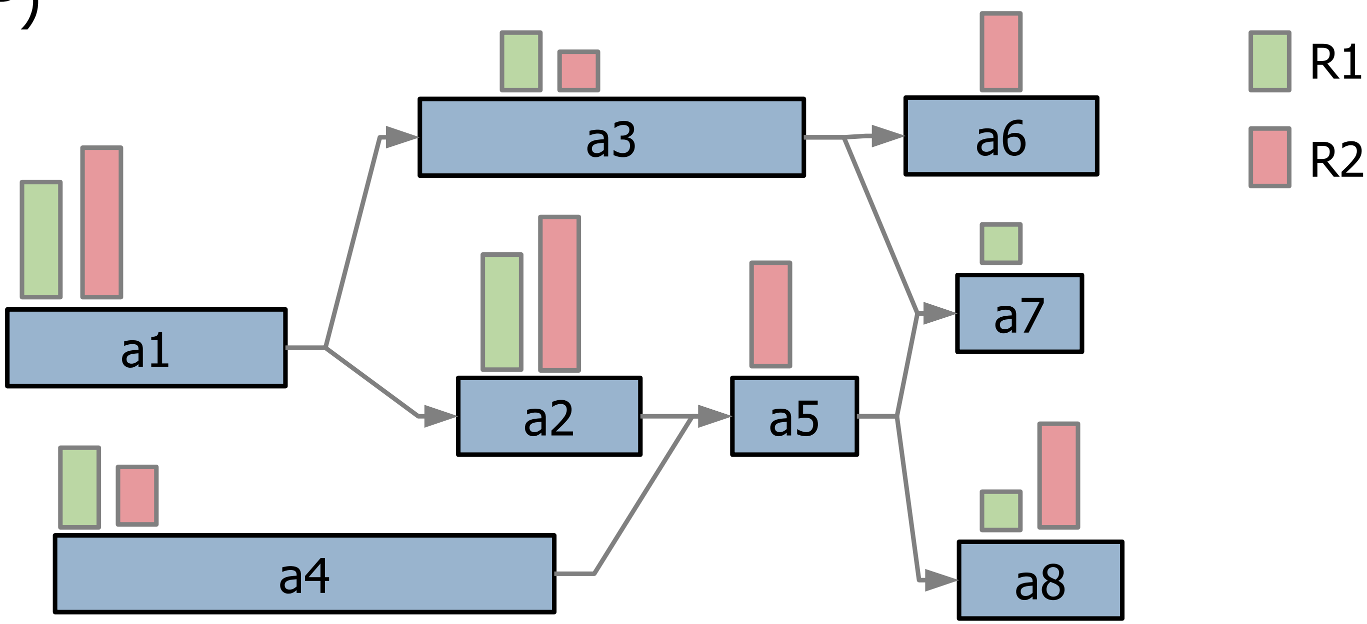


    - CP Optimizer model:



Concept: **cumul function**

- Example: Resource-Constrained Project Scheduling Problem (RCPSP)



- Minimization of project makespan

Concept: **cumul function**

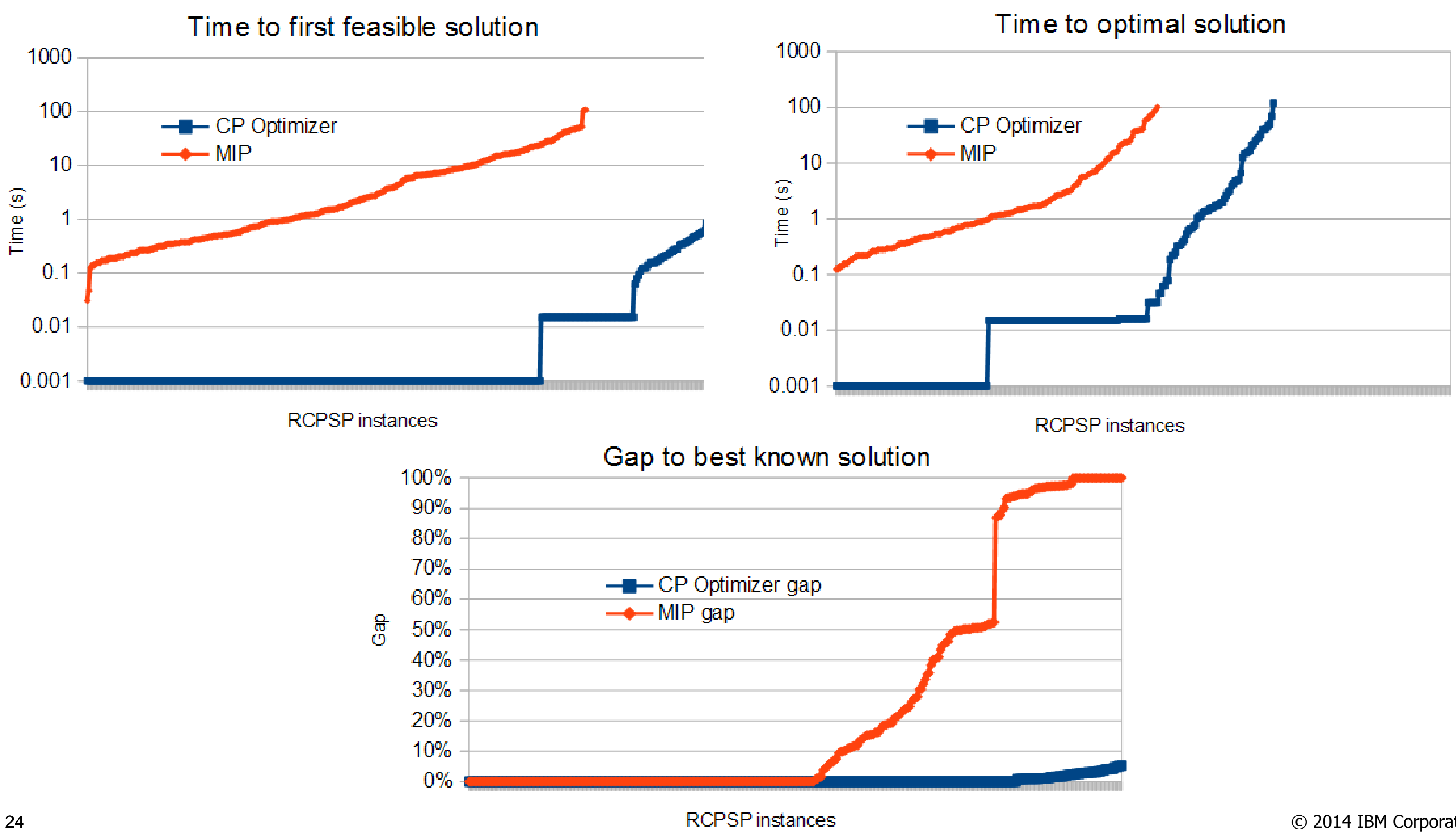
- CP Optimizer model for RCPSP:

```
dvar interval a[i in Tasks] size i.pt;

cumulFunction usage[r in Resources] =
  sum (i in Tasks: i.qty[r]>0) pulse(a[i], i.qty[r]);

minimize max(i in Tasks) endOf(a[i]);
subject to {
  forall (r in Resources)
    usage[r] <= Capacity[r];
  forall (i in Tasks, j in i.succs)
    endBeforeStart(a[i], a[<j>]);
}
```

- Comparison of this CP Optimizer model vs a time-indexed MIP model on a set of 300 classical small RCPSP instances (30-120 tasks) + 40 larger ones (900 tasks), time-limit: 2mn, 4 threads

Concept: **cumul function**

- Comparison of CP Optimizer and MIP performance on RCPSP

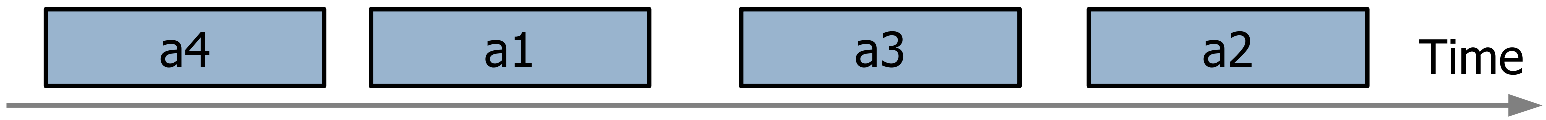


Concept: **sequence variable**

- What for?
  - modeling sequences of events
  - constraining the transitions between consecutive events (setup times/costs,...)
- Example:

```
dvar sequence s in all(i in Tasks) a[i]
noOverlap(s)
```

- Properties:
  - A value for the sequence variable is a total order on the present interval variables. E.g. a4 → a1 → a3 → a2
  - A constraint noOverlap states that intervals of the sequence do not overlap and follows the total order of the sequence



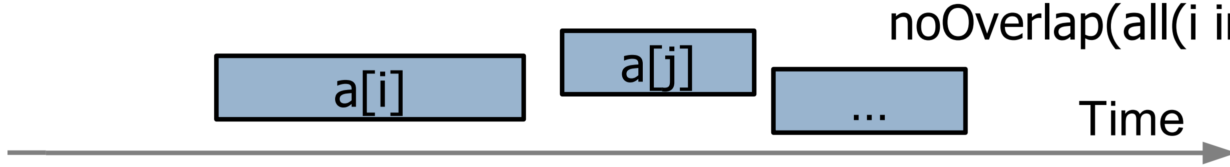
Concept: **sequence variable**

- Properties:
  - Complexity is **independent of the time scale**
  - CP Optimizer is able to reason **globally** over a sequence variable
  - **Avoid quadratic models** over each pair (i,j) of intervals in the sequence
- Compare:
  - Quadratic disjunctive MIP formulation with big-Ms:

```
b[i][j]∈{0,1}: b[i][j]=1 iff a[i] before a[j]
end[i] <= start[j] + M*(1-b[i][j])
end[j] <= start[i] + M*b[i][j]
```

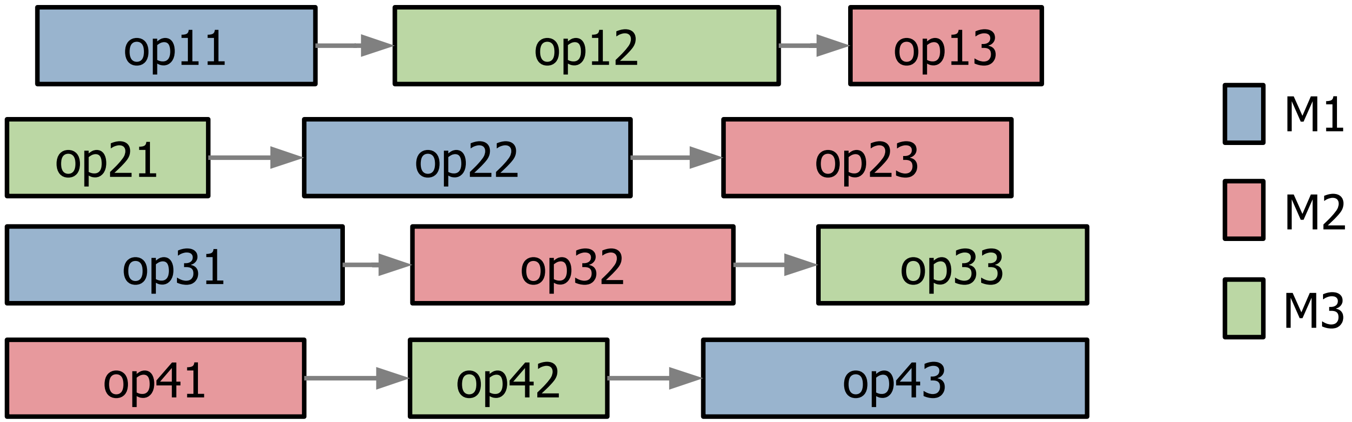
  - CP Optimizer model:

noOverlap(all(i in …) a[i])



Concept: **sequence variable**

- Example: Job-shop Scheduling Problem



- Minimization of makespan

Concept: **sequence variable**

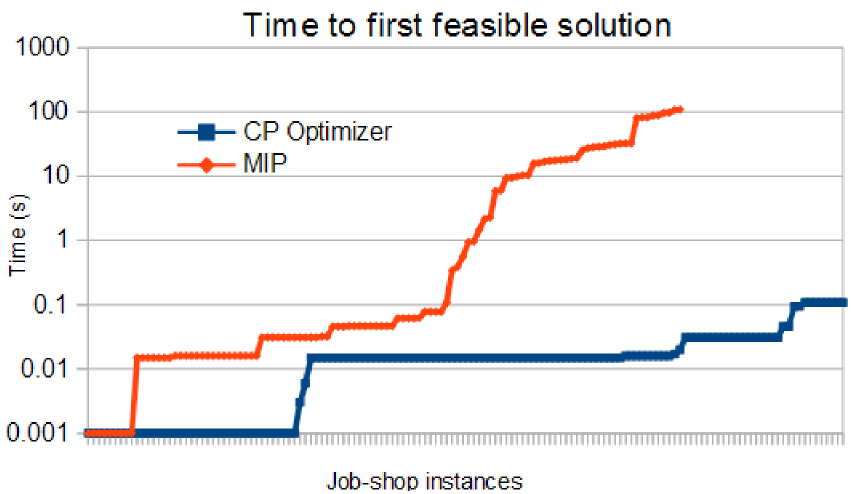
- CP Optimizer model for Job-shop:

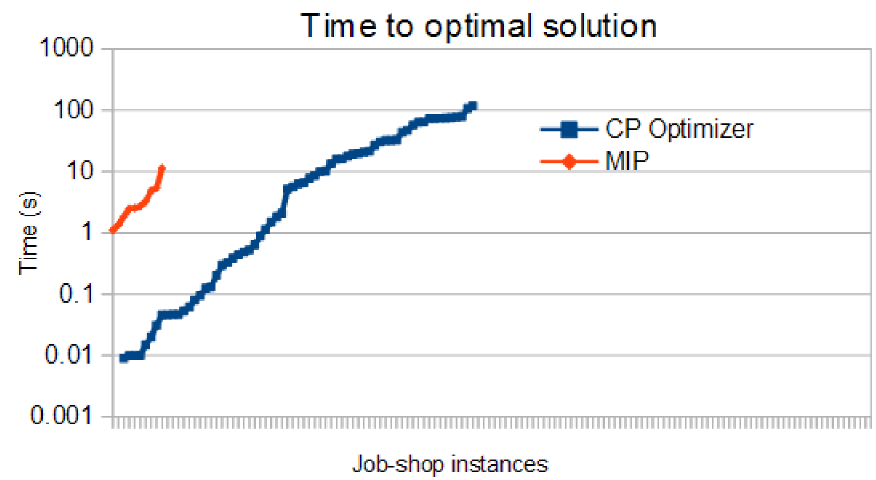
```
dvar interval op[j in Jobs][p in Pos] size Ops[j][p].pt;
dvar sequence mchs[m in Mchs] in
  all(j in Jobs, p in Pos : Ops[j][p].mch == m) op[j][p];

minimize max(j in Jobs) endOf(op[j][nbPos-1]);
subject to {
  forall (m in Mchs)
    noOverlap(mchs[m]);
  forall (j in Jobs, p in 1..nbPos-1)
    endBeforeStart(op[j][p-1], op[j][p]);
}
```

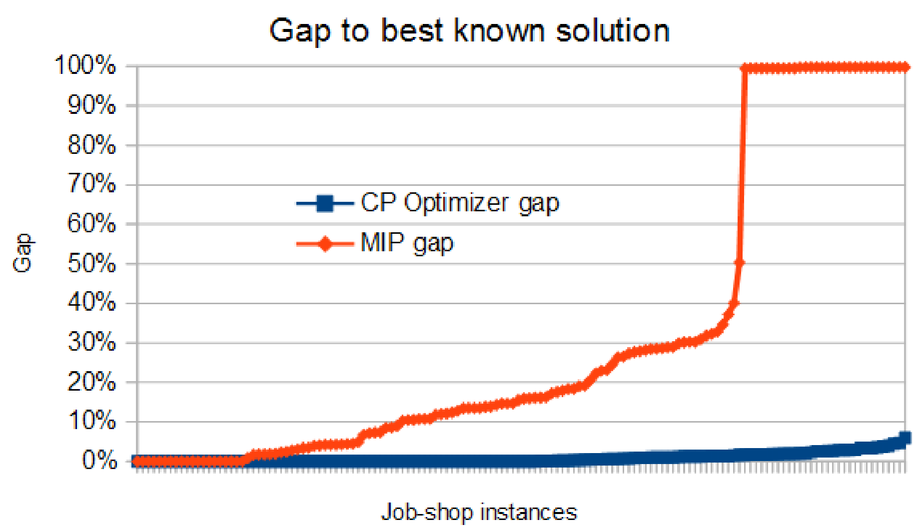
- Comparison of this CP Optimizer model vs a disjunctive MIP formulation on a set of 140 classical Job-shop instances (50-2000 tasks), time-limit: 2mn, 4 threads

Concept: **sequence variable**

- Comparison of CP Optimizer and MIP performance on Job-Shop

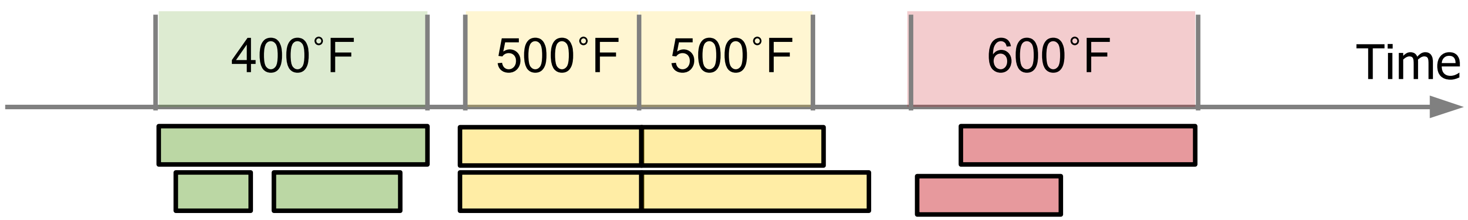






Concept: **state function**

- What for?
  - Modeling evolution of a state variable over time
  - Modeling incompatibility relations between activities overlapping a given date t
  - Synchronizing start/end value of compatible tasks (batching)
- Example

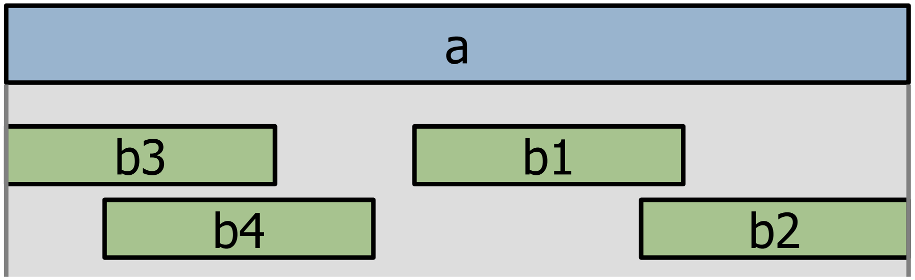


- Properties
  - Complexity is **independent of the time scale**
  - CP Optimizer is able to reason **globally** over a state function
  - **Avoid quadratic models** over each pair (i,j) of incompatible intervals on the state function

Concept: **span constraint**

- What for?
  - –Modeling task → sub-task decomposition
  - –Modeling immobilization time of a resource
- Example

```
span(a,[b1,...,bn])
```

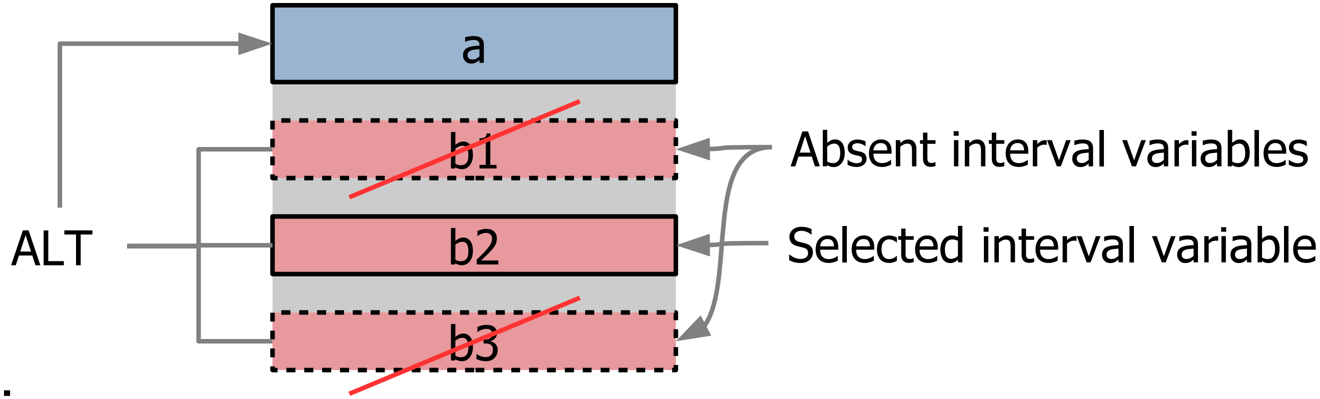


- Properties
  - –The constraint of course handles optional interval variables

Concept: **alternative constraint**

- What for?
  - Modeling alternative resource/modes/recipes
  - In general modeling a discrete selection in the schedule
- Example

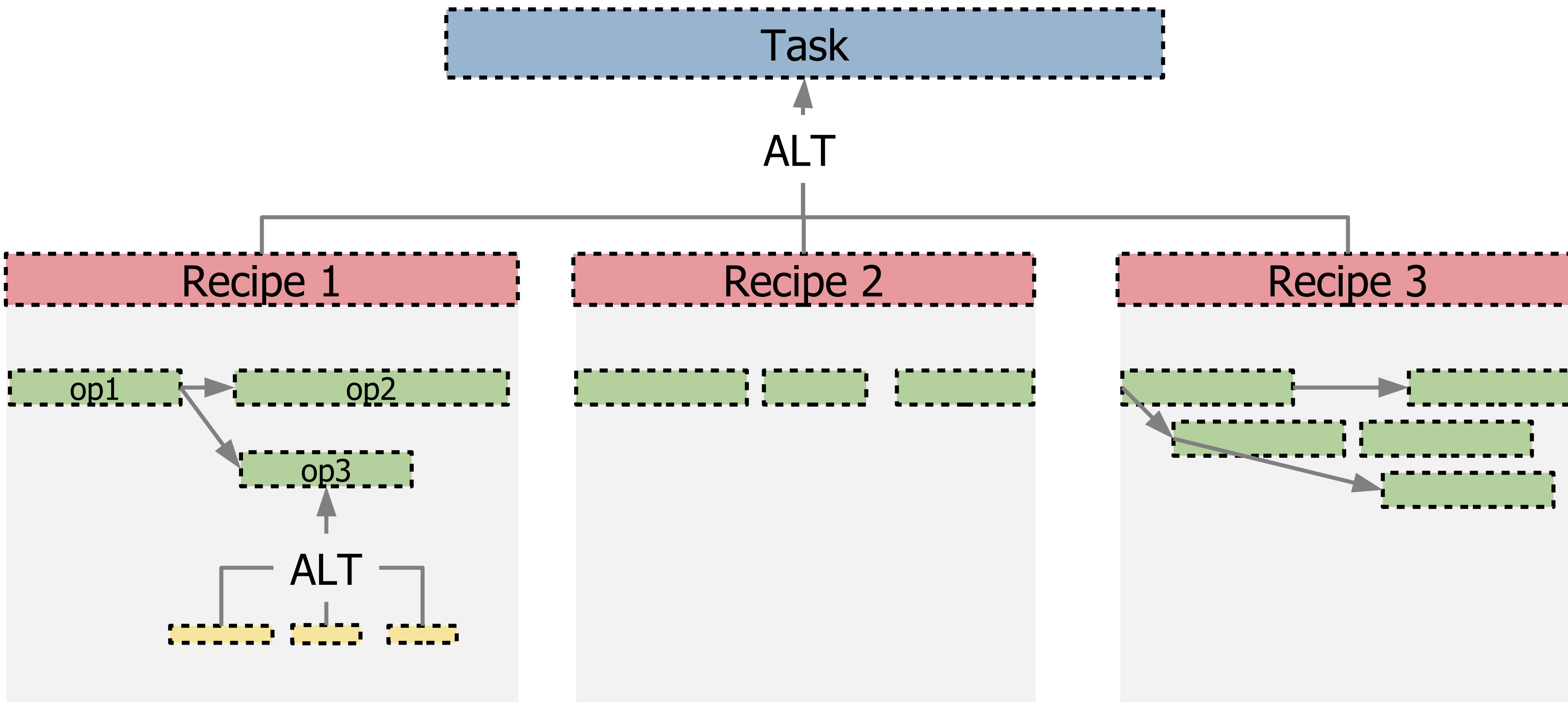
```
alternative(a,[b1,...,bn])
```



- Properties
  - Conditional reasoning allows a strong pruning on the alternative master interval variable *a*
  - Master interval variable *a* can of course be optional

Concept: **span/alternative constraint**

- What for?
  - Modeling Work Breakdown Structure
- Example



Agenda

- **IBM ILOG CP Optimizer** for **Scheduling**
- Scheduling **concepts** in CP Optimizer
- **Tips** and **Tools**
- **Q&A**

Tips and Tools

- A measure of complexity of a model:
  - **Number of variables**
  - Number of groups of variables with different semantics
  - Number of variable types (integer variables / interval variables)
- Take advantage of the expressiveness of the CP Optimizer modeling language to compact the model by decreasing the above indicators
- In particular if in a scheduling model, you are creating 1 decision variable for each time point *t* you are usually in bad shape

→ Exploit the scheduling concepts to reason on time-lines (sequence variables, cumul functions, state functions) and avoid an explicit representation of time

Tips and Tools

- A measure of complexity of a model is the **number of constraints**
- If the number of constraints grows more than linearly with the number of variables or the size of variables domains, the model will probably not scale well
- Furthermore, if a set of many small constraints can be reformulated more compactly, this often leads to stronger inference in the engine
- Example: when you need to model activities (and more generally intervals of time) that -under some conditions- cannot overlap, think of using sequence variables, noOverlap constraints and/or state functions

Tips and Tools

- Be careful when using **composite constraints**
- In scheduling models, exploit optionality of interval variables
- Examples :


  - presenceOf(a)*endOf(a)+(1-presenceOf(a))*K   USE: endOf(a,K)
  - presenceOf(a) => (10<=startOf(a))   USE: 10<=startOf(a,10)

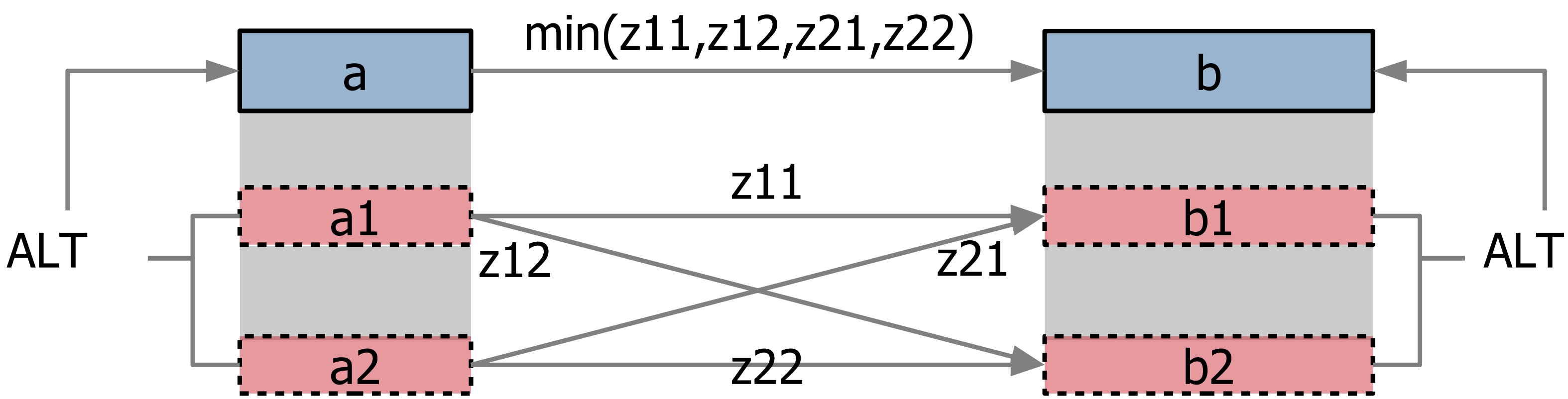
- Remarks:
  - **Exception**: Binary logical constraints on presence status (like presenceOf(a) => presenceOf(b)) are handled in a special (and efficient) way in the engine
  - Except for constraint "presenceOf", all constraints on scheduling constructs (interval and sequence variables, cumul and state functions) cannot be used in composite constraints

Tips and Tools

- Surrogate constraints **may** help to reduce the search space
- A surrogate constraint is a constraint that does not change the set of solutions (but may be useful for the search by allowing a better pruning of the search space)
- Some examples :
  - Redundant cumul function for alternative resources

Tips and Tools

- Surrogate constraints **may** help to reduce the search space
- A surrogate constraint is a constraint that does not change the set of solutions (but may be useful for the search by allowing a better pruning of the search space)
- Some examples :
  - –Redundant cumul function for alternative resources
  - –Redundant precedence constraints



Tips and Tools

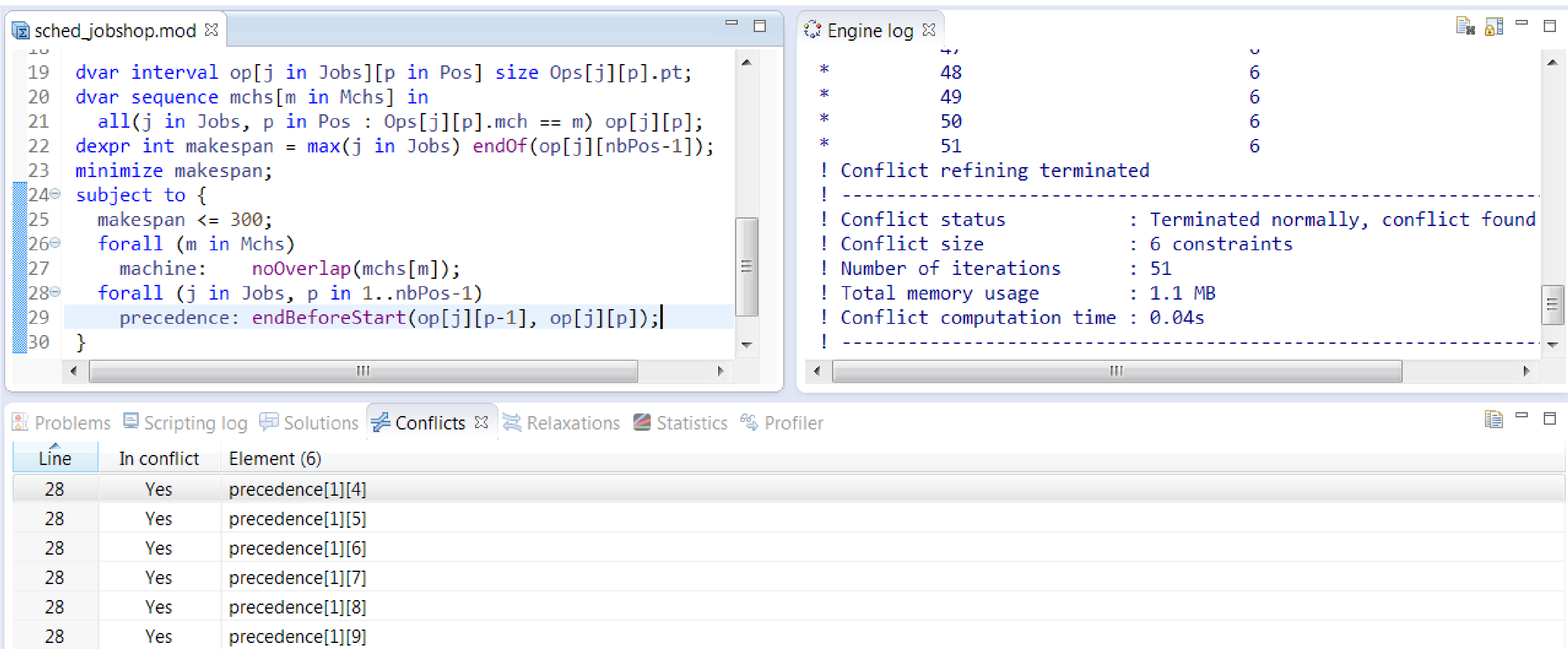
- Some classical patterns in scheduling models
  - Chain of optional interval variables
  - Optional intervals in paths of precedence constraints
  - Multilevel alternatives
  - Temporal alternatives
- *See bonus material at the end of the slide deck*

Tips and Tools

- Questions:
  - Why is my model infeasible ?
  - Search does not find any feasible solution, what should I do ?
  - Search does not converge fast enough, what should I do ?
  - Does my model improvement really help?

Tips and Tools

- Why is my model infeasible ?
  - –Conflict refiner → Fix/relax model



Tips and Tools

- The engine log is an essential tool for understanding how the engine behaves on a given model

```
! ----------------------------------------------------------------------
! Minimization problem - 110 variables, 100 constraints
! LogPeriod            = 100,000
! Workers              = 2
! Initial process time : 0.00s (0.00s extraction + 0.00s propagation)
!  . Log search space  : 664.4 (before), 664.4 (after)
!  . Memory usage      : 1.1 MB (before), 1.2 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*         1,062      739 0.01s         1             -
...
*          936     136k 3.69s          1             -
           936     300k          85    2    F    on op#0#9
! Time = 4.05s, Explored branches = 450,928, Memory usage = 7.7 MB
!          Best Branches  Non-fixed    W       Branch decision
*          930     166k 4.55s          1             -
           930     200k         110    1    F    on op#5#1
! ----------------------------------------------------------------------
! Search terminated normally, 17 solutions found.
! Best objective         : 930 (optimal - effective tol. is 0)
! Number of branches     : 641,352
! Number of fails        : 266,640
! Total memory usage     : 6.9 MB (6.4 MB CP Optimizer + 0.5 MB Concert)
! Time spent in solve    : 5.99s (5.99s engine + 0.00s extraction)
! Search speed (br. / s) : 107,070.5
! ----------------------------------------------------------------------
```

Problem characteristics

Tips and Tools

- The engine log is an essential tool for understanding how the engine behaves on a given model

```
! ----------------------------------------------------------------------------
! Minimization problem - 110 variables, 100 constraints
! LogPeriod            = 100,000
! Workers              = 2
! Initial process time : 0.00s (0.00s extraction + 0.00s propagation)
!  . Log search space  : 664.4 (before), 664.4 (after)
!  . Memory usage      : 1.1 MB (before), 1.2 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*         1,062      739 0.01s         1             -
...
*           936     136k 3.69s         1             -
            936     300k        85     2    F    on op#0#9
! Time = 4.05s, Explored branches = 450,928, Memory usage = 7.7 MB
!          Best Branches  Non-fixed    W       Branch decision
*           930     166k 4.55s         1             -
            930     200k       110     1    F    on op#5#1
! ----------------------------------------------------------------------------
! Search terminated normally, 17 solutions found.
! Best objective         : 930 (optimal - effective tol. is 0)
! Number of branches     : 641,352
! Number of fails        : 266,640
! Total memory usage     : 6.9 MB (6.4 MB CP Optimizer + 0.5 MB Concert)
! Time spent in solve    : 5.99s (5.99s engine + 0.00s extraction)
! Search speed (br. / s) : 107,070.5
! ----------------------------------------------------------------------------
```

Modified parameter values

Tips and Tools

- The engine log is an essential tool for understanding how the engine behaves on a given model

```
! ----------------------------------------------------------------------------
! Minimization problem - 110 variables, 100 constraints
! LogPeriod            = 100,000
! Workers              = 2
! Initial process time : 0.00s (0.00s extraction + 0.00s propagation)
!  . Log search space  : 664.4 (before), 664.4 (after)
!  . Memory usage      : 1.1 MB (before), 1.2 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*         1,062      739 0.01s         1             -
...
*           936     136k 3.69s         1             -
            936     300k          85   2    F    on op#0#9
! Time = 4.05s, Explored branches = 450,928, Memory usage = 7.7 MB
!          Best Branches  Non-fixed    W       Branch decision
*           930     166k 4.55s         1             -
            930     200k         110   1    F    on op#5#1
! ----------------------------------------------------------------------------
! Search terminated normally, 17 solutions found.
! Best objective         : 930 (optimal - effective tol. is 0)
! Number of branches     : 641,352
! Number of fails        : 266,640
! Total memory usage     : 6.9 MB (6.4 MB CP Optimizer + 0.5 MB Concert)
! Time spent in solve    : 5.99s (5.99s engine + 0.00s extraction)
! Search speed (br. / s) : 107,070.5
! ----------------------------------------------------------------------------
```

Root node information

Tips and Tools

- The engine log is an essential tool for understanding how the engine behaves on a given model

```
! ----------------------------------------------------------------------------
! Minimization problem - 110 variables, 100 constraints
! LogPeriod              = 100,000
! Workers                = 2
! Initial process time : 0.00s (0.00s extraction + 0.00s propagation)
!  . Log search space  : 664.4 (before), 664.4 (after)
!  . Memory usage      : 1.1 MB (before), 1.2 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*         1,062      739 0.01s         1             -
...
*           936     136k 3.69s         1             -
            936     300k          85   2    F    on op#0#9
! Time = 4.05s, Explored branches = 450,928, Memory usage = 7.7 MB
!          Best Branches  Non-fixed    W       Branch decision
*           930     166k 4.55s         1             -
            930     200k         110   1    F    on op#5#1
! ----------------------------------------------------------------------------
! Search terminated normally, 17 solutions found.
! Best objective         : 930 (optimal - effective tol. is 0)
! Number of branches     : 641,352
! Number of fails        : 266,640
! Total memory usage     : 6.9 MB (6.4 MB CP Optimizer + 0.5 MB Concert)
! Time spent in solve    : 5.99s (5.99s engine + 0.00s extraction)
! Search speed (br. / s) : 107,070.5
! ----------------------------------------------------------------------------
```

New incumbent solutions (time, worker)

Tips and Tools

- The engine log is an essential tool for understanding how the engine behaves on a given model

```
! ----------------------------------------------------------------------------
! Minimization problem - 110 variables, 100 constraints
! LogPeriod            = 100,000
! Workers              = 2
! Initial process time : 0.00s (0.00s extraction + 0.00s propagation)
!  . Log search space  : 664.4 (before), 664.4 (after)
!  . Memory usage      : 1.1 MB (before), 1.2 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*         1,062      739 0.01s         1             -
...
*           936     136k 3.69s         1             -
            936     300k          85   2    F    on op#0#9
! Time = 4.05s, Explored branches = 450,928, Memory usage = 7.7 MB
!          Best Branches  Non-fixed    W       Branch decision
*           930     166k 4.55s         1             -
            930     200k         110   1    F    on op#5#1
! ----------------------------------------------------------------------------
! Search terminated normally, 17 solutions found.
! Best objective         : 930 (optimal - effective tol. is 0)
! Number of branches     : 641,352
! Number of fails        : 266,640
! Total memory usage     : 6.9 MB (6.4 MB CP Optimizer + 0.5 MB Concert)
! Time spent in solve    : 5.99s (5.99s engine + 0.00s extraction)
! Search speed (br. / s) : 107,070.5
! ----------------------------------------------------------------------------
```

Periodical log with fail information, number of unfixed variables, current decision

Tips and Tools

- The engine log is an essential tool for understanding how the engine behaves on a given model

```
! ----------------------------------------------------------------------
! Minimization problem - 110 variables, 100 constraints
! LogPeriod            = 100,000
! Workers              = 2
! Initial process time : 0.00s (0.00s extraction + 0.00s propagation)
!  . Log search space  : 664.4 (before), 664.4 (after)
!  . Memory usage      : 1.1 MB (before), 1.2 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*         1,062      739 0.01s         1             -
...
*           936     136k 3.69s         1             -
            936     300k         85    2    F    on op#0#9
! Time = 4.05s, Explored branches = 450,928, Memory usage = 7.7 MB
!          Best Branches  Non-fixed    W       Branch decision
*           930     166k 4.55s         1             -
            930     200k        110    1    F    on op#5#1
! ----------------------------------------------------------------------
! Search terminated normally, 17 solutions found.
! Best objective         : 930 (optimal - effective tol. is 0)
! Number of branches     : 641,352
! Number of fails        : 266,640
! Total memory usage     : 6.9 MB (6.4 MB CP Optimizer + 0.5 MB Concert)
! Time spent in solve    : 5.99s (5.99s engine + 0.00s extraction)
! Search speed (br. / s) : 107,070.5
! ----------------------------------------------------------------------
```

Final information with solution status and search statistics

Tips and Tools

- Search does not find any feasible solution, what should I do ?
  - Identify which part of the model is hard:
    - Simplify problem by removing “easy” constraints
    - Search log helps (set LogPeriod=1, look where first fails occur)
  - Strengthen the model:
    - Be careful with composite constraints
    - Add surrogate constraints
    - Break symmetries
  - Increase inference levels in the engine

Tips and Tools

- Search does not find any feasible solution, what should I do ?
  - Try using search phases to first focus on the difficult decision variables
  - Solve the problem as two sub-problems:
    1. Finding an initial feasible solution (some difficult constraints can be relaxed and put in the objective function*), then
    2. Optimize starting from this solution. Consider using starting points.
  - Isolate the different sources of complexity in the model by decomposition and sequential resolutions. Consider using starting points

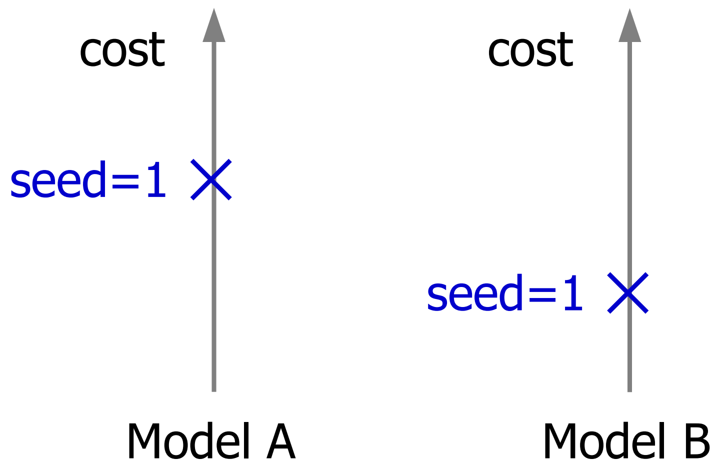
(*) CP Optimizer's concepts makes it easy to relax part of the model. E.g. make some interval variables *optional* and maximize number of present intervals, relax *deadlines* as *tardiness* costs

Tips and Tools

- Search does not converge fast enough, what should I do ?
  - Identify and try to **isolate the different sources of complexity** in your model, first focusing on bottleneck/important parts in a "simplified" model
  - Exploit results of previous solve as (1) additional constraints and/or (2) starting point and/or (3) heuristic (search phases)
  - Do not hesitate to use MIP models in some steps, exploit strengths of each technology (MIP/CP)
    - For instance CP is good at deciding activities **start/end times** under resource constraints when required resource quantities and activity durations are fixed or mostly depend on activity start/end dates
    - For problems with very variable **resource quantities** or **activity durations**, it makes sense to consider an initial MIP model that works on these variables (example: lot sizing)

Tips and Tools

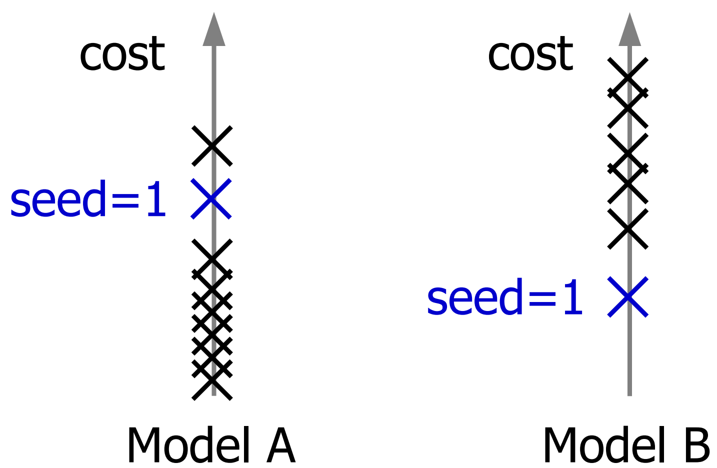
- Does my model improvement really help?
  - –CP Optimizer's search relies on a stochastic algorithm initialized by a random seed (parameter IloCP::RandomSeed)
  - –For a given model, the solution depends on the random seed
  - –Best solution cost at a given time limit:



- –Is model B better than model A ? Yes

Tips and Tools

- Does my model improvement really help?
  - CP Optimizer's search relies on a stochastic algorithm initialized by a random seed (parameter IloCP::RandomSeed)
  - For a given model, the solution depends on the random seed
  - Best solution cost at a given time limit:



Never evaluate a
model change on
a single instance/seed !

No!

- Is model B better than model A ? ~~Yes~~

Conclusion

- For scheduling problems, CP Optimizer extends the **Model&run** paradigm with some (**few**) general concepts (interval & sequence variables, cumul & state functions, …)
- These concepts can be used to model scheduling problems in a compact way that **avoids enumeration of time**. Typically, the size of a scheduling model in CP Optimizer grows linearly with the size of the data
- When combined together these concepts provide an expressive language capable of handling **complex** and **large** industrial scheduling problems
- The **automatic search** is **complete**, it exploits these concepts using an **efficient** and **robust** combination of techniques from scheduling theory, meta-heuristics, CP, MP

Agenda

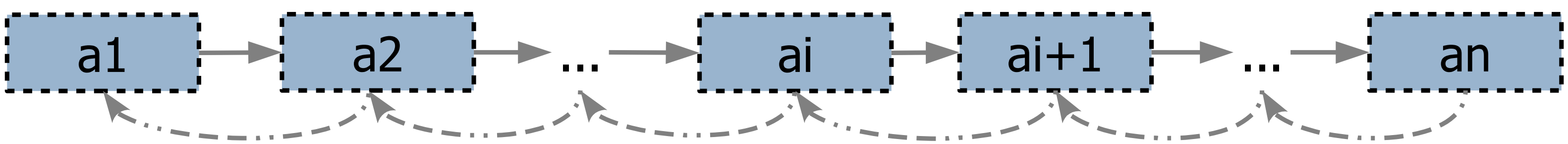
- **IBM ILOG CP Optimizer** for **Scheduling**
- Scheduling **concepts** in CP Optimizer
- **Tips** and **Tools**
- **Q&A**

Bonus material

- List of most useful parameters for search analysis:
  - Workers (=1)
  - LogPeriod (=1)
  - SearchType (=DepthFirst)
  - FailLimit, SolutionLimit (=1)
- List of most useful parameters for performance tuning:
  - TimeLimit
  - TemporalRelaxation (=Off)
  - Workers
  - SearchType (=Restart|MultiPoint)
  - Some inference levels:
    - NoOverlapInferenceLevel
    - CumulFunctionInferenceLevel

Bonus material: classical patterns in scheduling models

- Chain of optional interval variables
  - Use cases: any phenomenon that is associated with a set of consecutive time intervals whose number is not known a-priori
  - Examples: partially preemptive activities, intervals during which a resource is in use, …
- Model is a chain of precedence and implication constraints between optional interval variables

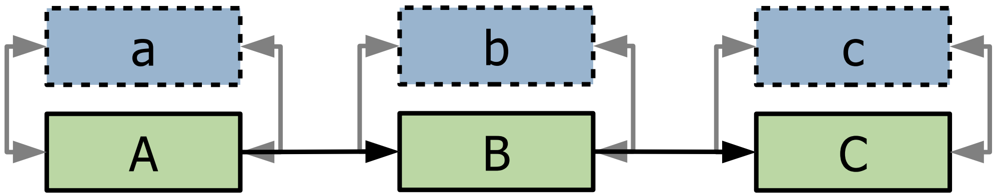



```
endBeforeStart(a[i],a[i+1]);
presenceOf(a[i+1]) => presenceOf(a[i]);
```

- This ensures that only the first k intervals will be present where k is an implicit decision of the problem

Bonus material: classical patterns in scheduling models

- Optional intervals in paths of precedence constraints
  - Use case: a precedence graph with optional activities such that path of precedence constraints going through absent activities should still be considered
  - Example: a → b → c, precedence a → c should be considered even if b is absent
  - Direct model using endBeforeStart(a,b), endBeforeStart(b,c) does not work as both constraints are inactive if b is absent
- Idea 1: Add endBeforeStart(x,y) constraints of the transitive closure. Issue: may add a quadratic number of constraints
- Idea 2: Use additional non-optional interval variables A, B, C of free length synchronized with a, b, c (startAtStart, endAtEnd)

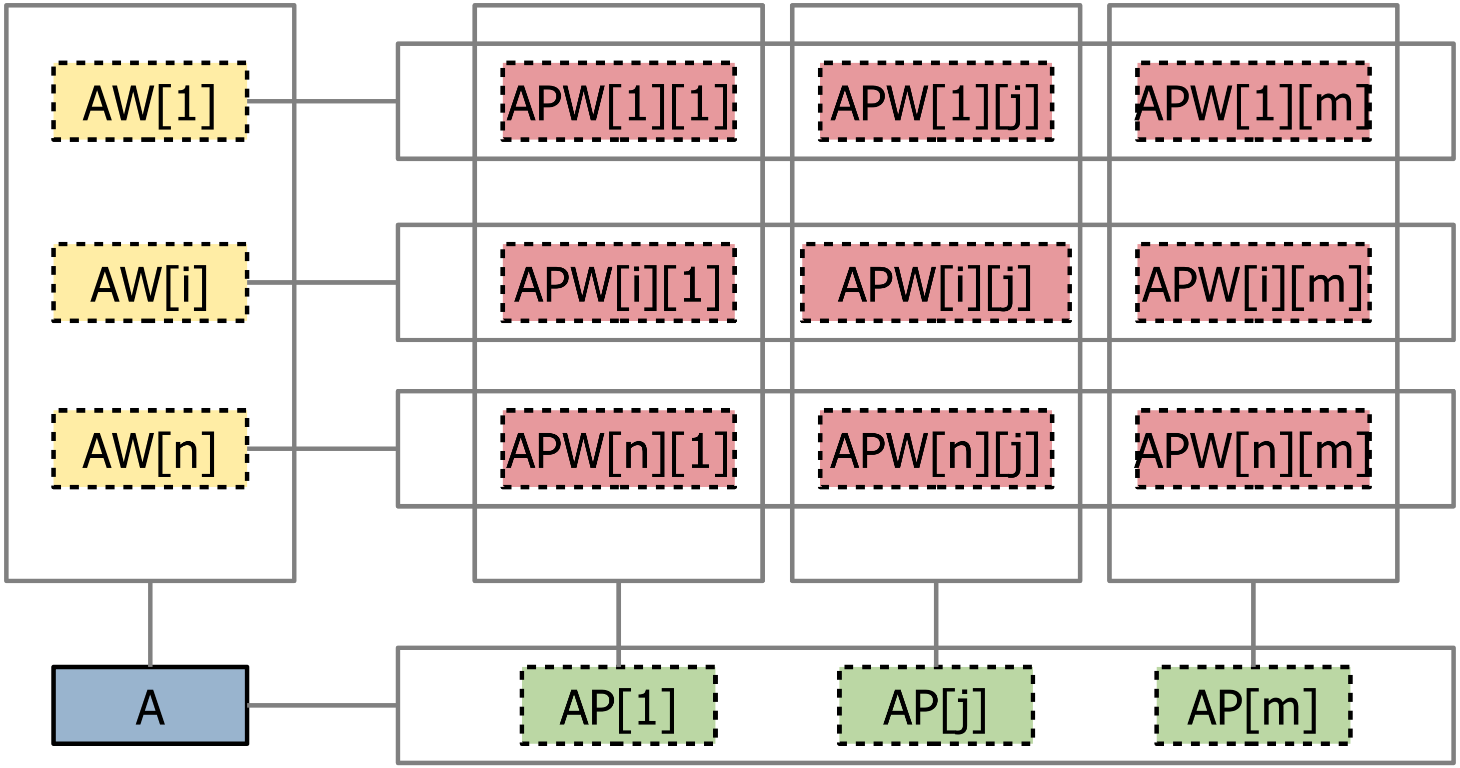


Bonus material: classical patterns in scheduling models

- Multi-level alternatives
  - Use case: an activity needs to select an element in a Cartesian product
  - Example: an activity A needs 1 worker Wi among n to be executed at 1 given position Pj among m. Duration depends both on the worker Wi and the position Pj. Specific constraints need to be posted on all activities executing at a position Pj (e.g. space limitation) while other constraints need to be posted on all activities executed by a given worker Wi (e.g. noOverlap)
- Idea: create one optional interval variable AW[i] for activity A executing on worker Wi and one optional interval variable AP[j] for activity A executing a position Pj

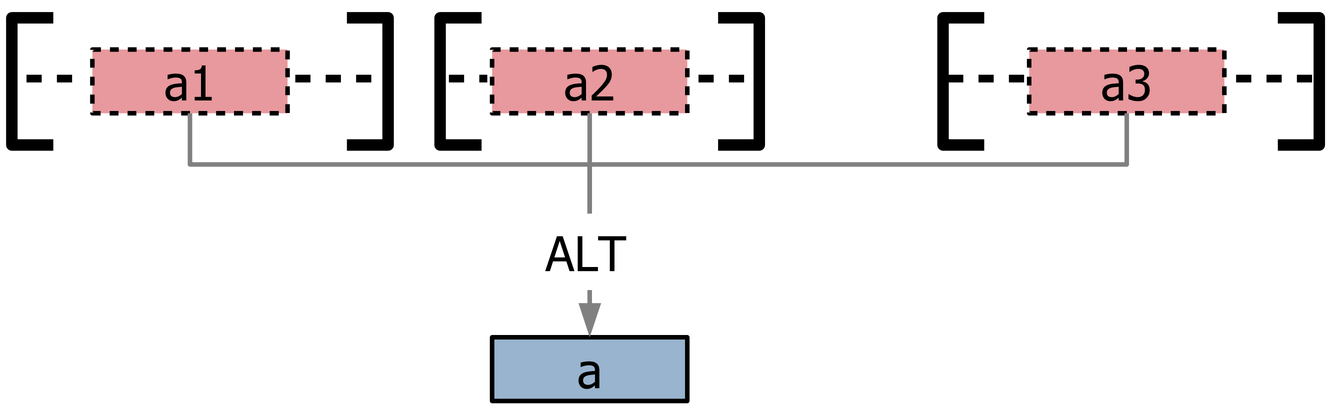
Bonus material: classical patterns in scheduling models

- Multi-level alternatives
- Idea: create one optional interval variable AW[i] for activity A executing on worker Wi and one optional interval variable AP[j] for activity A executing at position Pj



Bonus material: classical patterns in scheduling models

- Temporal alternatives
  - Alternative constraints are not restricted to model disjunctions of resources/modes/recipes, they can also be used to represent temporal disjunction
  - Use-case: an activity must execute during a disjunctive set of possible time-windows



- Alternative models that directly work on interval variable *a* are possible (use cumulFunction or forbidExtent). They will be lighter (less variables) but will propagate less.

Bonus material

- Technical papers related with scheduling in CP Optimizer
  - P. Laborie, J. Rogerie. *Temporal Linear Relaxation in IBM ILOG CP Optimizer*. Proc. MISTA 2013. Ghent, Belgium. Aug. 2013.
  - P. Laborie, J. Rogerie , P. Shaw, P. Vilím. *Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources*. Proc. FLAIRS 2009.
  - P. Vilím. *Max Energy Filtering Algorithm for Discrete Cumulative Resources*. Proc. CPAIOR 2009. p294-308.
  - P. Laborie. *IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems*. Proc. CPAIOR 2009. p148-162.
  - P. Vilím. *Edge Finding Filtering Algorithm for Discrete Cumulative Resources in O(kn log n)*. Proc. CP 2009. p802-816.
  - P. Laborie, J. Rogerie. *Reasoning with Conditional Time-intervals*. Proc. FLAIRS 2008, p555-560.
  - P. Laborie, D. Godard. *Self-Adapting Large Neighborhood Search: Application to single-mode scheduling problems*. Proc. MISTA 2007. Paris, France. Aug. 2007.
  - D. Godard, P. Laborie, W. Nuijten. *Randomized Large Neighborhood Search for Cumulative Scheduling*. Proc. ICAPS-05. Monterey, California (USA). June 2005.