Philippe Laborie
09/03/2015

Solving Scheduling Problems with CP Optimizer (for CPLEX Users)

Please Note: This document contains IBM confidential material

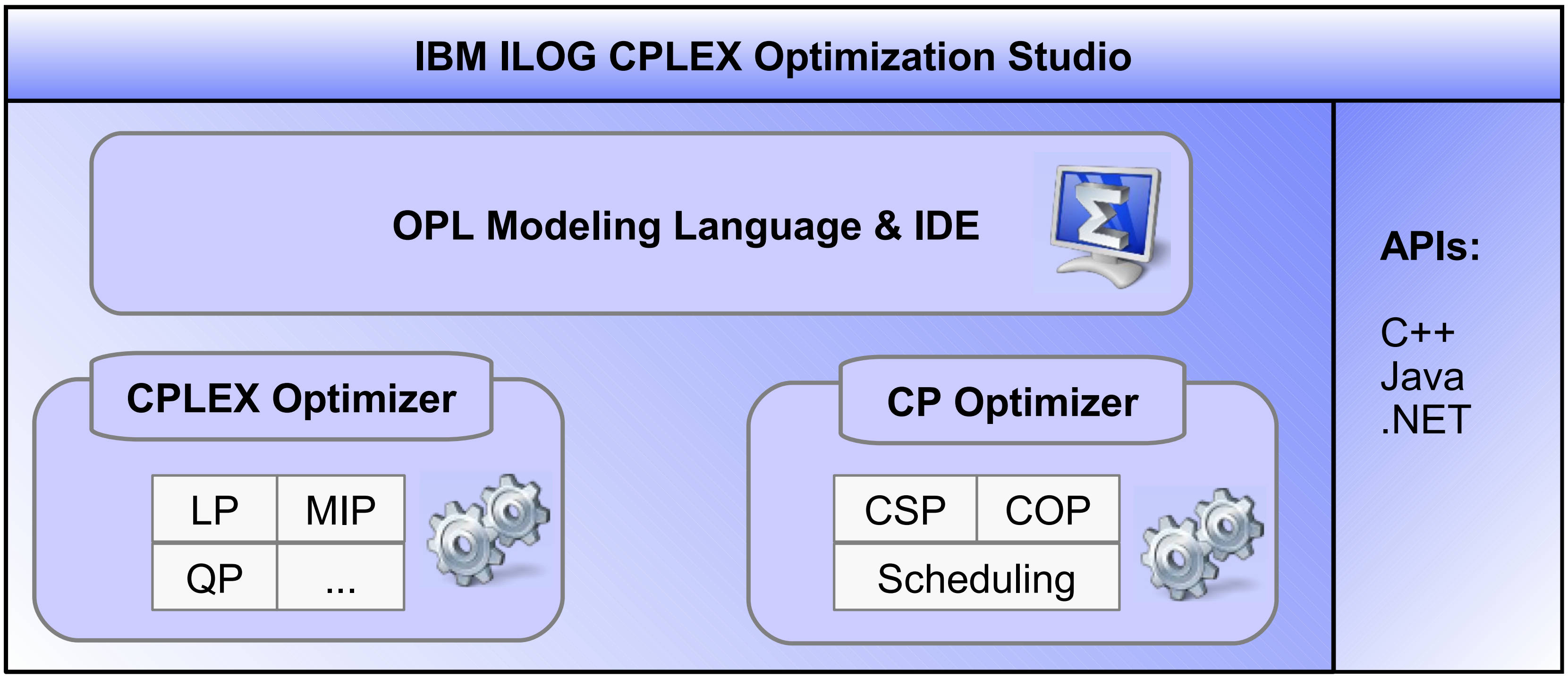
IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.
Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM® benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

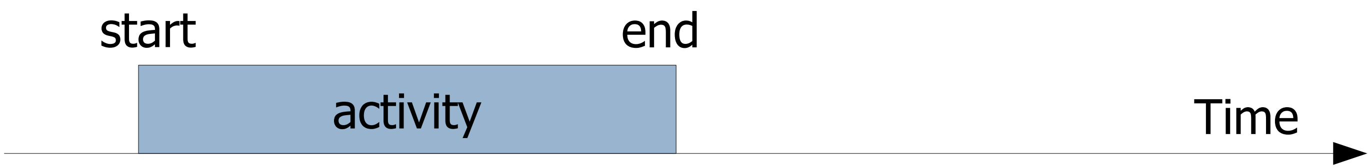
- **Introduction:**
  - **CP Optimizer** a component of CPLEX Optimization Studio
  - **Scheduling**
  - **Example**: Resource Constrained Project Scheduling Problem
- Scheduling **concepts** in CP Optimizer
- **Automatic search** in CP Optimizer
- Some features for **accelerating model development**
- **Q&A**

CP Optimizer, a component of CPLEX Optimization Studio



Scheduling

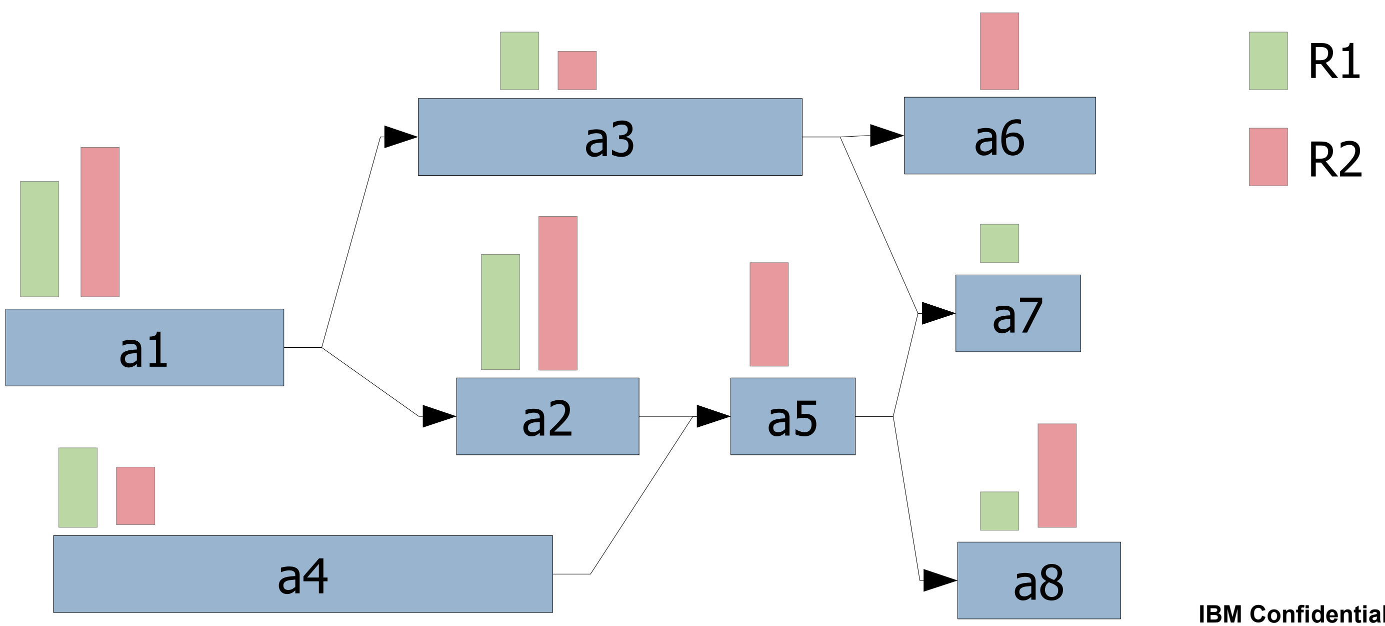
- Scheduling consist of assigning **starting** and **completion times** to a set of activities while satisfying different types of constraints (resource availability, precedence relationships, …) and optimizing some criteria (minimizing tardiness, …)



- 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**, **optional** activities …)

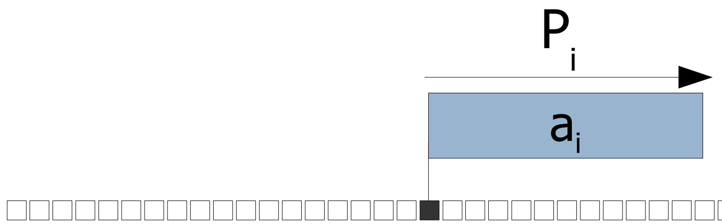
Example: Resource Constrained Project Scheduling Problem

- RCPSP: a very classical academical scheduling problem
  - Tasks a_i with fixed processing time P_i
  - Precedence constraints
  - Discrete resources with limited instantaneous capacity R_k
  - Tasks require some quantity of discrete resources
  - Objective is to minimize the schedule makespan



Example: Resource Constrained Project Scheduling Problem

- RCPSP: Standard time-indexed MIP formulation



$x_{it} \in \{0,1\}$

Standard RCPSP (DT: Discrete Time)

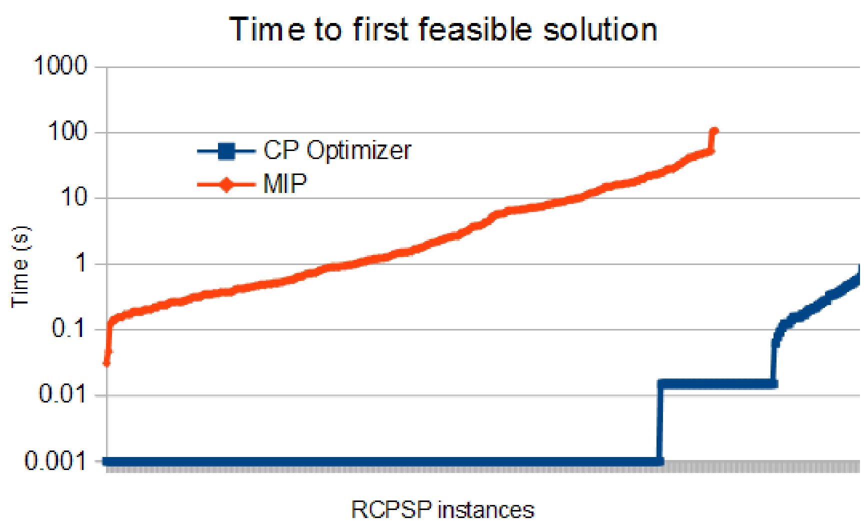
$$
\begin{aligned}
&\text{minimize} \sum_{t \in H} t x_{nt} && \\
&\sum_{t \in H} x_{it} = 1 && \forall i \in \mathcal{A} \\
&\sum_{t \in H} t x_{it} + P_i \leq \sum_{t \in H} t x_{jt} && \forall (i,j) \in \mathcal{P} \\
&\sum_{i \in A, t \leq \tau < t + P_i} Q_{ik} x_{it} \leq R_k && \forall \tau \in H, \forall k \in \mathcal{R} \\
&x_{it} \in \{0,1\} && \forall i \in \mathcal{A}, \forall t \in H
\end{aligned}
$$

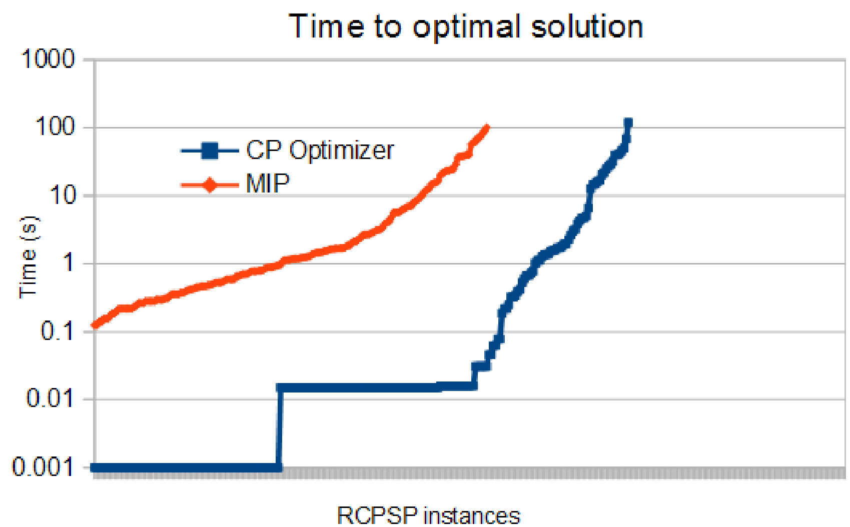
Example: Resource Constrained Project Scheduling Problem

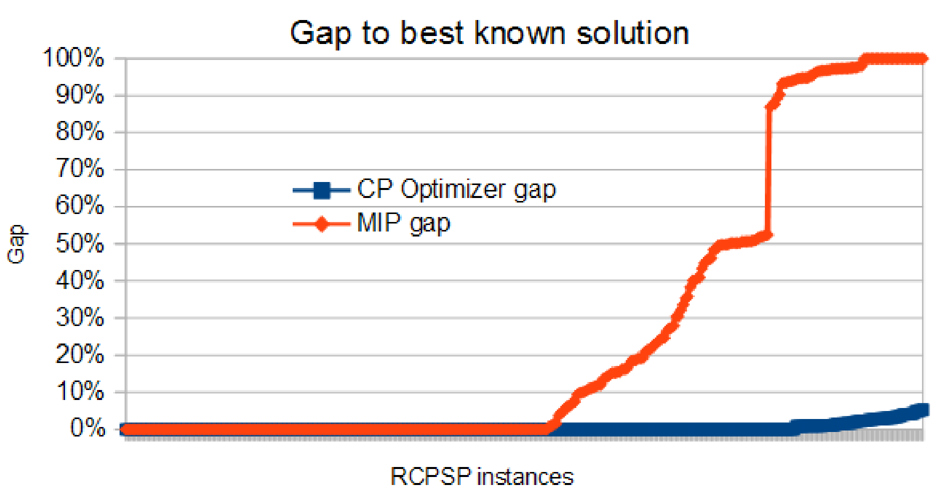
- Comparison of this time-indexed MIP formulation against a simple CP Optimizer model (see later) on a set of:
  - 300 classical **small** RCPSP instances (30-120 tasks) +
  - 40 slightly **more realistic** larger ones (900 tasks)
  - time-limit: 2mn, 4 threads
- Note: industrial scheduling problems are often **larger**, typically several 1.000 tasks (we handled up to 1.000.000 tasks in an RCPSP-like scheduling model)

Example: Resource Constrained Project Scheduling Problem

- Comparison of CP Optimizer and MIP performance on RCPSP







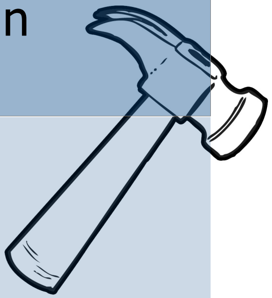
Example: Resource Constrained Project Scheduling Problem

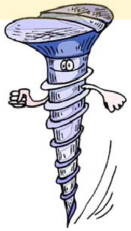
- Some pain points of the MIP approach (also hold for more general scheduling problems) :
  - 1. **Huge number of decision variables**:
    - Time-indexed formulations grow in $O(n.H)$
    - Other formulations (event-based, precedence-based for disjunctive resources) grow in $O(n^2)$
  - 2. The **time dimension is not exploited**. Example: building a first feasible solution by fixing the tasks chronologically is trivial. The MIP engine does not see that.
  - 3. **Global constraints** like resource capacities are split into a number of small individual constraints ($\forall t \in H$) whereas what happens at t+1 is highly correlated with what happens at t.

CP Optimizer for Scheduling

- Modeling and solving scheduling problems

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





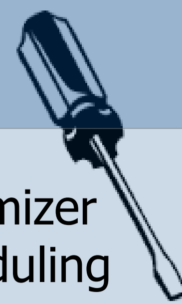
CP Optimizer for Scheduling

- Modeling and solving scheduling problems

Model & run paradigm

MIP (CPLEX)

CP Optimizer for Scheduling



User-implemented solution

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 (provides optimality proofs)
    - Combines tree search, large neighborhood search, meta-heuristics, relaxations, learning, ...

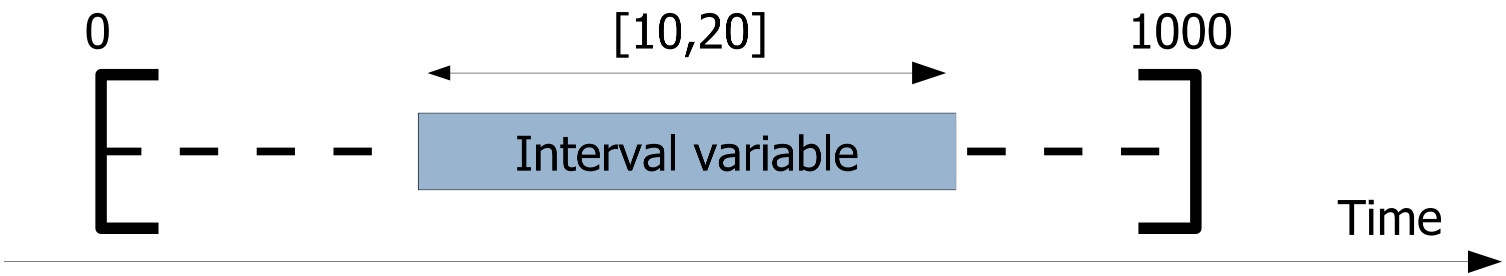
Agenda

- **Introduction:**
  - **CP Optimizer** a component of CPLEX Optimization Studio
  - **Scheduling**
  - **Example**: Resource Constrained Project Scheduling Problem
- Scheduling **concepts** in CP Optimizer
- **Automatic search** in CP Optimizer
- Some features for **accelerating model development**
- **Q&A**

Concept: **interval variable**

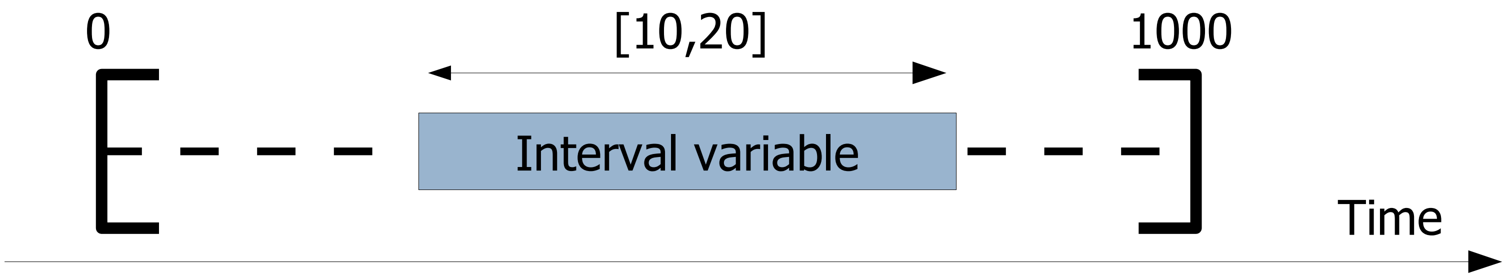
- What for?
  - modeling an interval of time during which a particular property holds (an activity executes, a resource is idle, a tank must be non-empty, …)
- Example:

```
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:

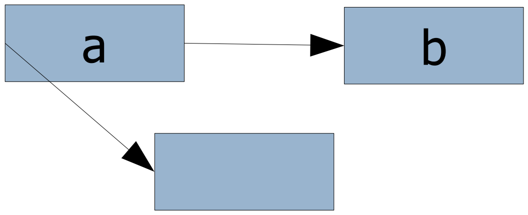
```
dvar interval x optional in 0..1000
                size in 10..20
```

- 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

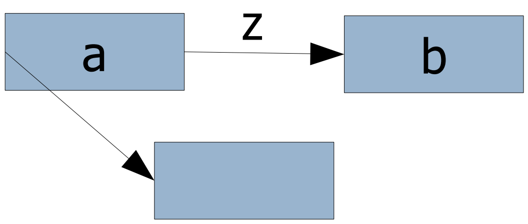
- Example:




```
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**

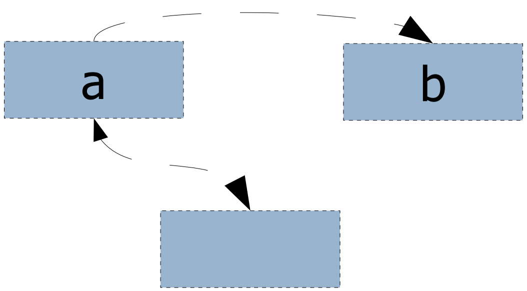
- Properties
  - Semantic of the constraints handles optionality (as for all constraints in CP Optimizer).

    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 (2-SAT) 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**

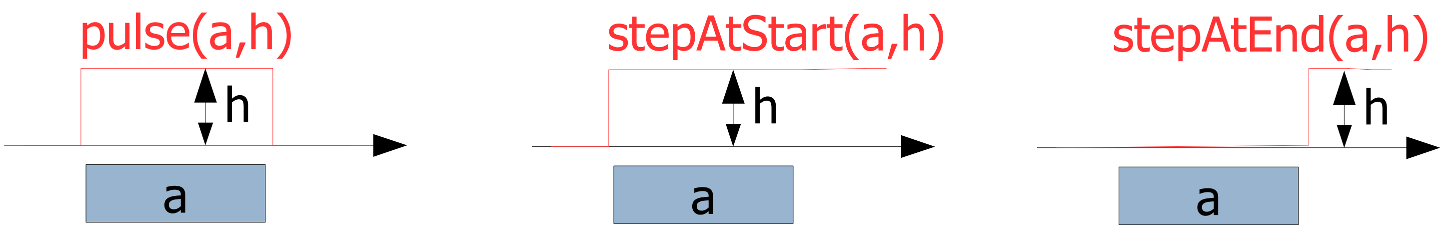
- What for?
  - Evaluating a characteristic of an interval variable (start, end, size) for use in the objective function or in a constraint
- Example:
  - endOf(a, ABSVAL) takes value ABSVAL if interval *a* is absent otherwise it takes value *e* if *a* is present and *a=[s,e)*
  - Typical makespan expression: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

Concept: **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**

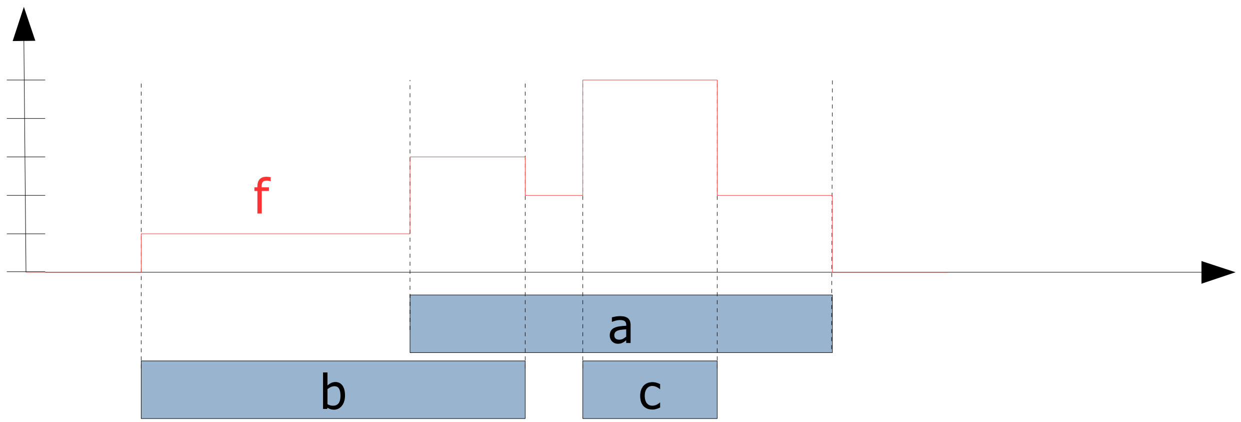
- Cumul functions are built from atomic functions



- 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**

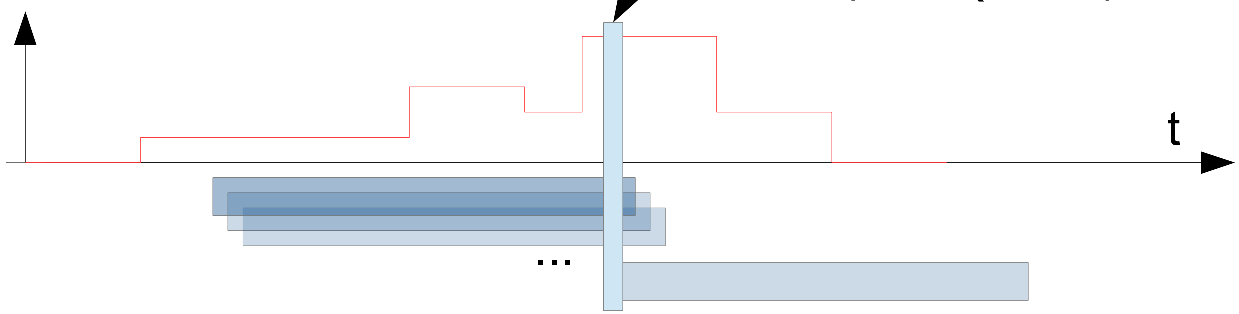
- Properties:
  - Complexity of cumul functions is **independent of the time scale**
  - CP Optimizer is able to reason globally over cumul functions

Concept: **cumul function**

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

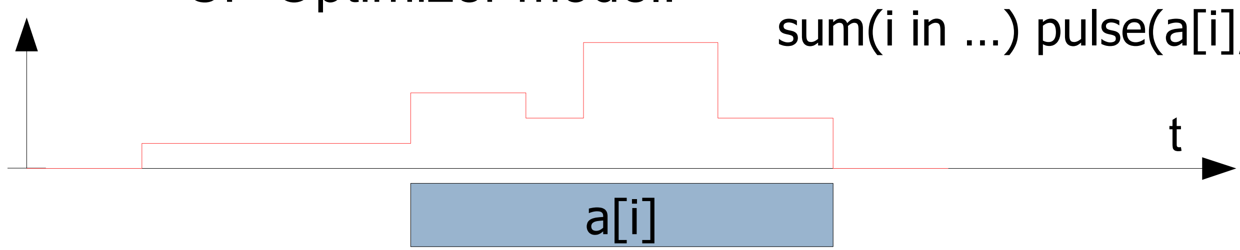
      `x[i][t]∈{0,1}: x[i][t]=1 iff a[i] starts at date t`

      ∀t, sum(i in ..., t' in ...) Q[i]*x[i][t']≤ C



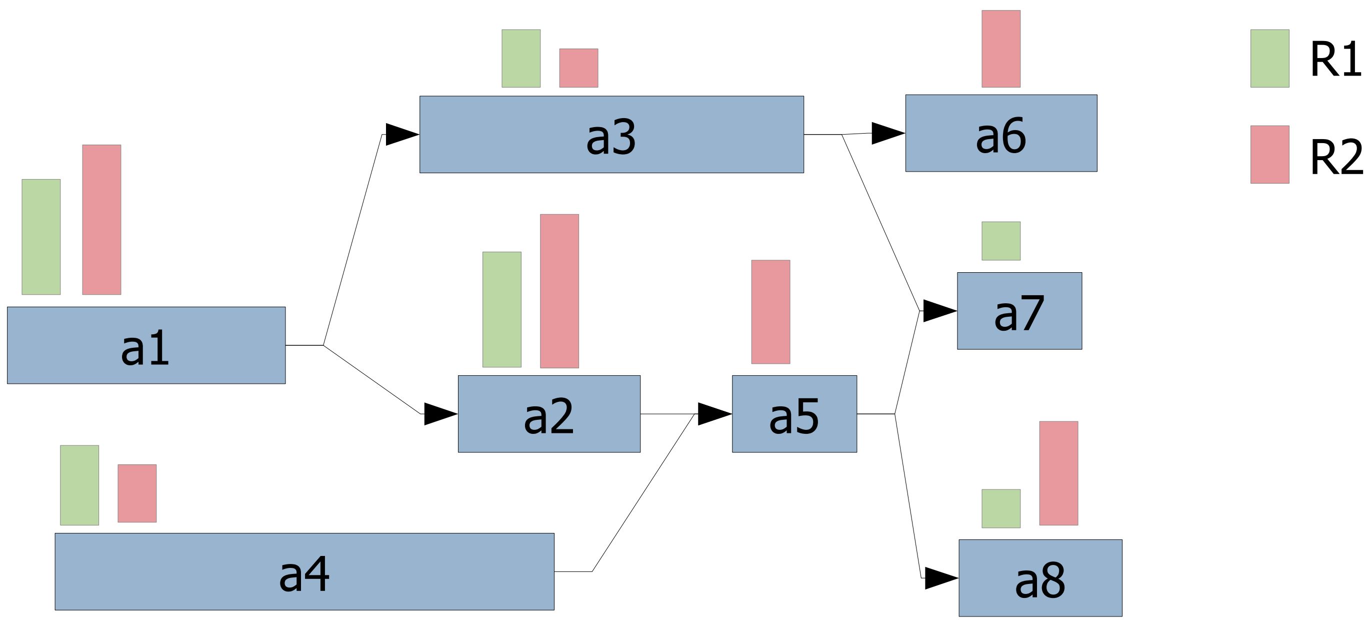
    - CP Optimizer model:

      sum(i in ...) pulse(a[i],Q[i]) ≤ C



Concept: **cumul function**

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



- Minimization of project makespan

Concept: **cumul function**

- CP Optimizer model for RCPSP:

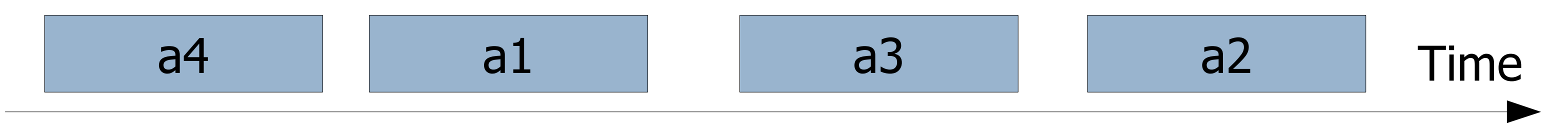
```
dvar interval a[i in Tasks] size i.pt;

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>]);
}
```

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



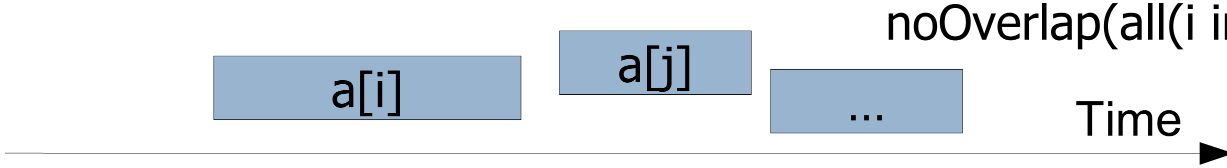
Concept: **sequence variable**

- 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]
```

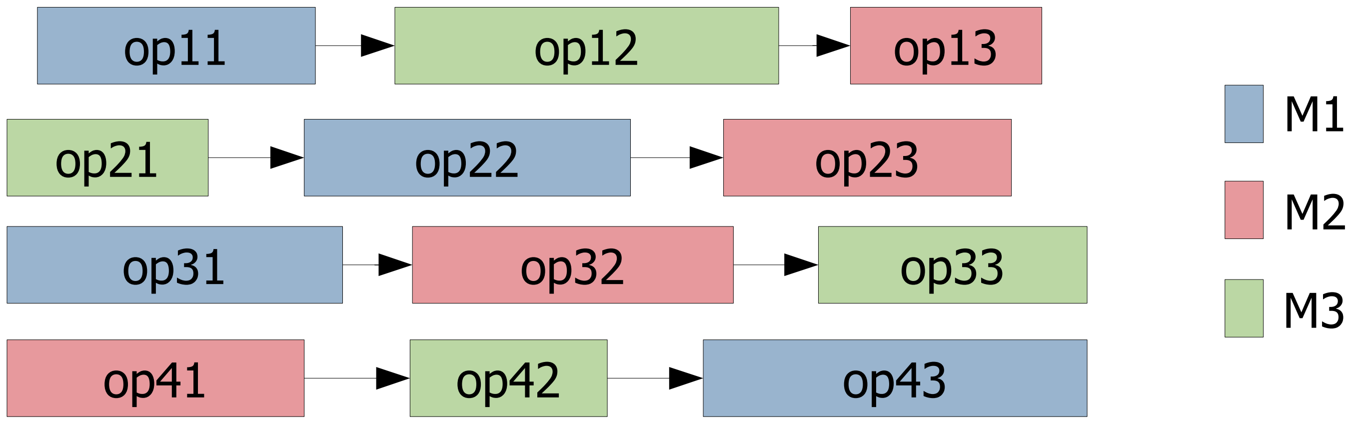
  - CP Optimizer model:

noOverlap(all(i in …) a[i])



Concept: **sequence variable**

- Example: Job-shop Scheduling Problem



- Minimization of makespan

Concept: **sequence variable**

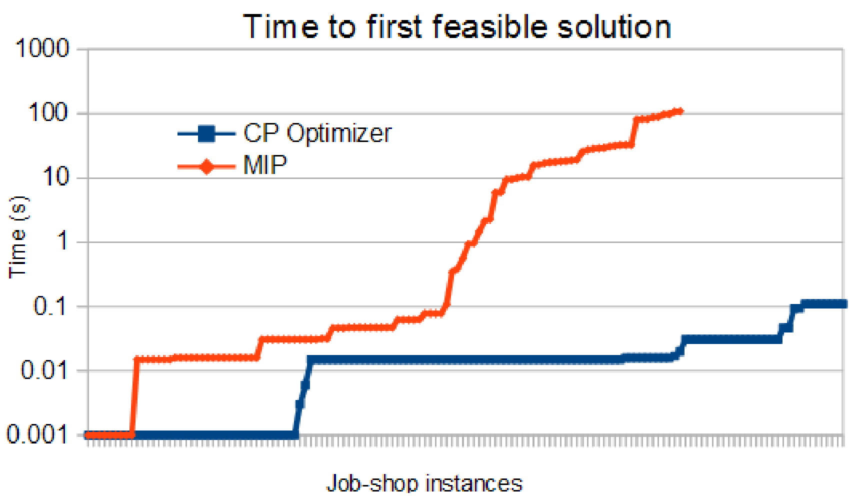
- CP Optimizer model for Job-shop:

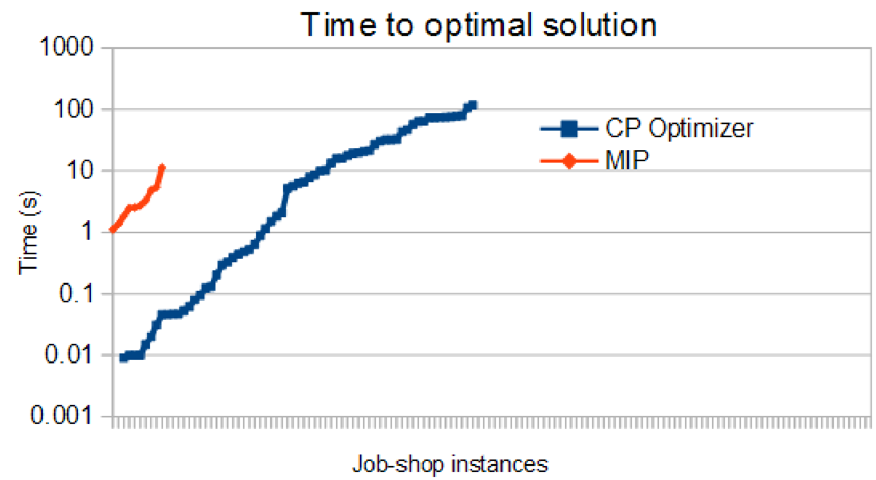
```
dvar interval op[j in Jobs][p in Pos] size Ops[j][p].pt;
dvar sequence mchs[m in Mchs] in
  all(j in Jobs, p in Pos : Ops[j][p].mch == m) op[j][p];

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]);
}
```

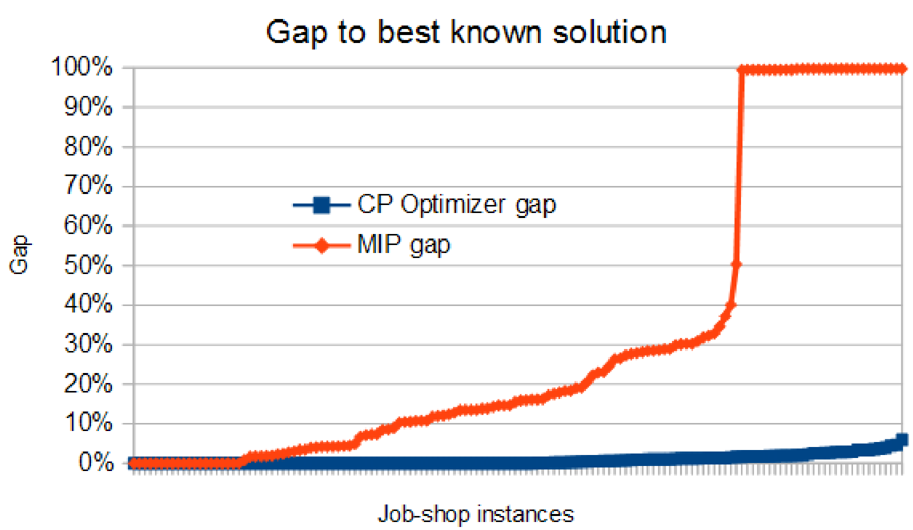
- Comparison of this CP Optimizer model vs a disjunctive MIP formulation on a set of 140 classical Job-shop instances (50-2000 tasks), time-limit: 2mn, 4 threads

Concept: **sequence variable**

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



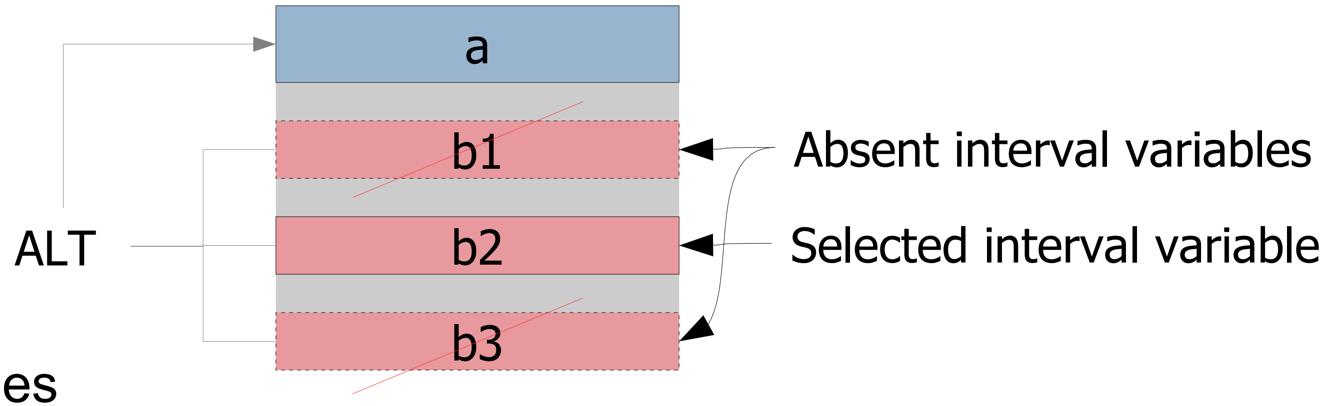




Concept: **alternative constraint**

- 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

Integer variables and expressions in CP Optimizer

- CP Optimizer is not only about scheduling !
- … and complex scheduling problems may involve other decision variables than interval variables !
- CP Optimizer supports integer decision variables and a large panel of expressions/constraints

<table>
<tr><th>Variables</th><th>Expressions</th><th>Constraints</th></tr>
<tr>
<td>Variables are discrete integer

Domains can be specified as a range [1..50] or as a set of values {1, 3, 5, 7, 9}

dvar int x in 1..50</td>
<td>Expressions can be integer or floating-point, for example 0.37 * y is allowed

Basic arithmetic and more complex operators (min, max, log, pow etc.) are supported

Relational expressions can be treated as 0-1 expressions.
e.g. z = (y < t)

Special expressions:
y == a[x]
t == count(b, 3)
t == cond ? x : y</td>
<td>Rich set of constraints

Standard relational constraints (==, !=, <, >, <=, >=)

Logical combinators (&&, ||, !, =>)

Specialized (global) constraints
allDifferent(b)
allowedAssignments(b, tuples)
forbiddenAssignments(b, tuples)
pack(load, container, size)
lexicographic(b, c)
inverse(b, c)</td>
</tr>
</table>

Integer variables and expressions in CP Optimizer

- Model can mix integer and interval variables

```
dvar int qty[i in 1..n] in …;
dvar interval task[i in 1..n] in 0..H;

maximize sum(i in 1..n)
   qty[i] * pow( H-endOf(task[i]), 1.025 );
```

Agenda

- **Introduction:**
  - **CP Optimizer** a component of CPLEX Optimization Studio
  - **Scheduling**
  - **Example**: Resource Constrained Project Scheduling Problem
- Scheduling **concepts** in CP Optimizer
- **Automatic search** in CP Optimizer
- Some features for **accelerating model development**
- **Q&A**

Automatic Search

- Automatic search is invoked by a **single call** to:
  - IloCP::solve()
- Automatic search is **complete**:
  - If the problem is unfeasible, it will prove unfeasibility
  - If the problem is feasible, it will find an optimal solution and prove optimality
- Note that especially for scheduling problems, **proofs are very challenging** (some scheduling problems with as few as 50 tasks are still open)
- Once a first feasible solution has been found, the search can be interrupted at **any time** to get the best incumbent solution
- Search **parameters** are available to tune and limit the search (e.g. time limit)

Automatic Search

- Core CP techniques used as a building block:
  - Tree search (Depth First)
  - Constraint propagation
- Other techniques used:
  - Model presolve
  - Restarting techniques
  - No-good learning
  - Randomization
  - Impact-based branching
  - Opportunistic probing
  - Dominance rules
  - LP-assisted heuristics
  - Large Neighborhood Search
  - Algorithms portfolios and Machine learning
  - Evolutionary algorithms

Automatic Search

- Performance of the automatic search of CP Optimizer is continuously monitored on a set of more than 90 different scheduling benchmarks (e.g. RCPSP is just one of these benchmarks)

Campaign: PerfWorker1

	COS122	COS123	COS124	COS125	COS1251	COS126
COS123	1.16	-	-	-	-	-
COS124	1.19	1.02	-	-	-	-
COS125	1.48	1.28	1.25	-	-	-
COS1251	1.53	1.32	1.29	1.02	-	-
COS126	3.09	2.65	2.58	2.07	2.07	-
COS12610	3.03	2.57	2.51	1.99	1.99	0.98

- Performance of automatic search has been compared with the problem specific state-of-the-art methods in a number of scheduling benchmarks

Automatic Search

MISTA 2007

- Some results:

Problem type	Benchmark	Problem size	Reference UB	MRD	# Imp. UBs / # Instances
Trolley	[41]	230-460	[19]	−11.8%	15/15
Hybrid flow-shop	[35]	200-1000	[35]	−8.8%	19/20
Job-shop w/ E/T	[3]	30-200	[3]	−5.6%	32/48
Air traffic management	[19]	2000	[19]	−4.0%	1/1
Flow-shop w/ E/T	[27]	30-400	[14]	−3.0%	4/12
Max. quality RCPSP	[33]	30	[33]	−2.3%	NA/3600
Cumulative job-shop	[28]	150-675	[17]	−0.3%	27/86
Single proc. tardiness	[20]	200-500	[20]	0.2%	0/20
Semiconductor testing	[30]	400	[30]	0.2%	7/18
RCPSP w/ E/T	[42]	30-50	[42]	0.4%	15/60
Open-shop	[9, 40, 18]	64-400	[15, 7, 25]	0.9%	0/28
RCPSP	[23]	120	Best PSPLIB	1.6%	0/600[3]
Shop w/ setup times	[10]	50-200	[2]	2.3%	0/15
Parallel machine w/ E/T	[29]	8-200	[4]	2.6%	2/52
Job-shop	[1, 39, 43, 40]	100-500	Best OR-Lib	2.8%	0/33
Air land	[5]	10-50	[5]	3.4%	0/8
Flow-shop w/ buffers	[40]	100-500	[8]	3.6%	12/30
Flow-shop	[40]	100-500	Best OR-Lib	5.9%	0/22
Aircraft assembly	[16]	575	[13]	8.7%	0/1
Single machine w/ E/T	[11, 37, 29]	8-500	[38]	9.8%	1/100
Common due-date	[6]	100-200	[36]	14.7%	0/20

Table 1: Results of SA-LNS on 21 scheduling benchmarks

Automatic Search

CP-AI-OR 2009

Some results:

– Problem #1: Flow-shop with earliness/tardiness cost
– Problem #2: Oversubscribed Satellite Scheduling problem
– Problem #3: Personal tasks scheduling

	OPL Model size	CPO Automatic search (no parameter tuning) vs. state-of-the-art
#1	20 lines	Competitive with state of the art (GA, LNS)
#2	15 lines	Number of unscheduled tasks decreased by 5%
#3	42 lines	Finds solution to more instances Solution quality increased by 12.5%

Automatic Search

CP-AI-OR 2015

Some results:

Benchmark set	Number of instances	Lower bound improvements	Upper bound improvements	Closed instances
JobShop	48	40	3	15
JobShopOperators	222	107	215	208
FlexibleJobShop	107	67	39	74
RCPSP	472	52	1	0
RCPSPMax	58	51	23	1
MultiModeRCPSP (j30)	552	No reference	3	535
MultiModeRCPSPMax	85	84	77	85

Table 1. Results summary

Agenda

Introduction:

- **CP Optimizer** a component of CPLEX Optimization Studio
- **Scheduling**
- **Example**: Resource Constrained Project Scheduling Problem

Scheduling **concepts** in CP Optimizer

Automatic search in CP Optimizer

Some features for **accelerating model development**

Q&A

Some features for accelerating model development

CP Optimizer provides similar features as CPLEX:

- Search log
- Conflict refiner
- Warm start
- Human-readable input/output file format (new in 12.6.1)

```
// --------------------------------------------------------------------------
// IBM ILOG CP Optimizer model export file
// Effective workers: 8
// --------------------------------------------------------------------------
// ------ Interval-related variables: ---------------------------------------
"itvs(0)(5)" = intervalVar(size=2);
"itvs(1)(5)" = intervalVar(size=3);
...
"mchs(0)" = sequenceVar(["itvs(0)(4)", "itvs(1)(1)", "itvs(2)(3)", "itvs(3)(5)", "itvs(4)(4)", "itvs(5)(2)"]);
"mchs(1)" = sequenceVar(["itvs(0)(1)", "itvs(1)(0)", "itvs(2)(2)", "itvs(3)(2)", "itvs(4)(0)", "itvs(5)(4)"]);
...
// ------ Objective: --------------------------------------------------------
minimize(max([endOf("itvs(0)(5)"), endOf("itvs(1)(5)"), ...]));
// ------ Constraints: ------------------------------------------------------
noOverlap("mchs(0)");
noOverlap("mchs(1)");
...
endBeforeStart("itvs(0)(0)", "itvs(0)(1)");
endBeforeStart("itvs(0)(1)", "itvs(0)(2)");
...
```

Some features for accelerating model development

- CPLEX and CP Optimizer engines are available in the same CPLEX Optimization Studio ecosystem and with the same "look&feel"

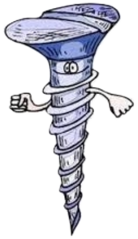
		CPLEX	**CP Optimizer**
Interfaces		OPL, C++, Java, .NET, C, Python	OPL, C++, Java, .NET
Model	Decision variables	int, float	int, interval
	Expressions	linear, quadratic	arithmetic, log, pow, … relational, a[x], count,...
	Constraints	range	relational, logical, specialized, scheduling
Search	Search parameters	✓	✓
	Warm start	✓	✓
	Multi-core //	✓	✓
Tools	Search log	✓	✓
	I/O format	.lp, .mps, ...	.cpo
	Conflict refiner	✓	✓

Some features for accelerating model development

- CPLEX and CP Optimizer engines are available in the same CPLEX Optimization Studio ecosystem and with the same “look&feel”
- A few differences still:
  - In CP, lower bounds are not directly available.
    Rationale: CP uses Depth-First instead of Best-First search, LB becomes known only at the very end of the search space traversal.
  - CP Optimizer allows for the implementation of user-defined constraints and search tree exploration (in C++ only).
    This can be compared to CPLEX user cuts and callbacks.
  - No feas-opt feature available so far for CP Optimizer.
    Rationale: due to the large typology of constraint, the violation degree of a constraint is highly dependent on the context. But optionality of interval variables makes it easy to model oversubscribed problems.

Some features for accelerating model development

- CPLEX and CP Optimizer engines are available in the same CPLEX Optimization Studio ecosystem and with the same "look&feel"
- Real problems are very seldom pure and monolithic

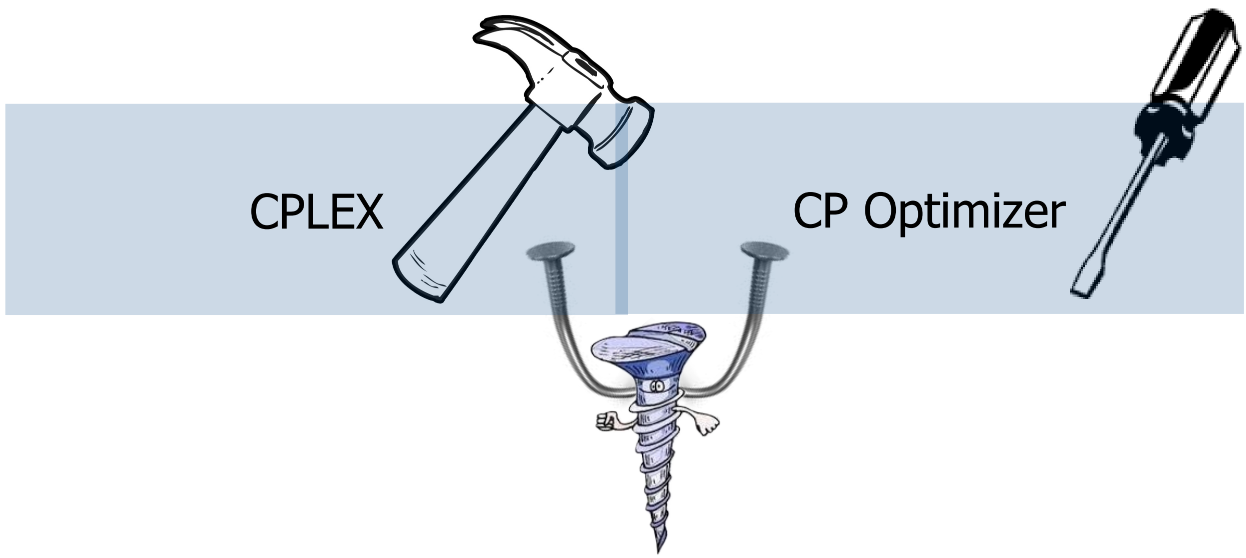


Some features for accelerating model development

- CPLEX and CP Optimizer engines are available in the same CPLEX Optimization Studio ecosystem and with the same “look&feel”
- Real problems are very seldom pure and monolithic

Some features for accelerating model development

- CPLEX and CP Optimizer engines are available in the same CPLEX Optimization Studio ecosystem and with the same "look&feel"
- Real problems are very seldom pure and monolithic



- CPLEX and CP Optimizer are **complementary**:
  - Sequential models (planning → detailed scheduling)
  - Use MIP to compute LB (relaxed model) or guidelines
  - Logic-based Benders decomposition, column generation

Agenda

Introduction:

- **CP Optimizer** a component of CPLEX Optimization Studio
- **Scheduling**
- **Example**: Resource Constrained Project Scheduling Problem

Scheduling **concepts** in CP Optimizer

Automatic search in CP Optimizer

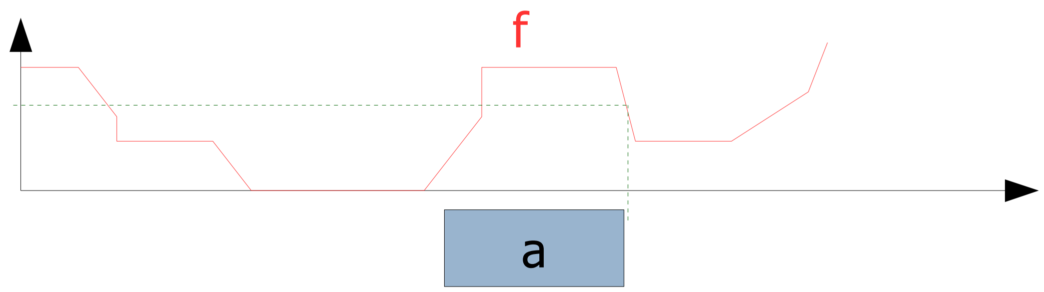
Some features for **accelerating model development**

Q&A

Backup

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)*

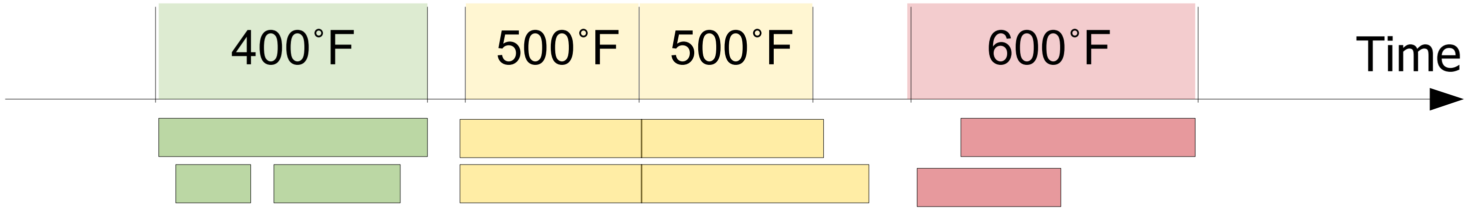


Property:

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

Concept: **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

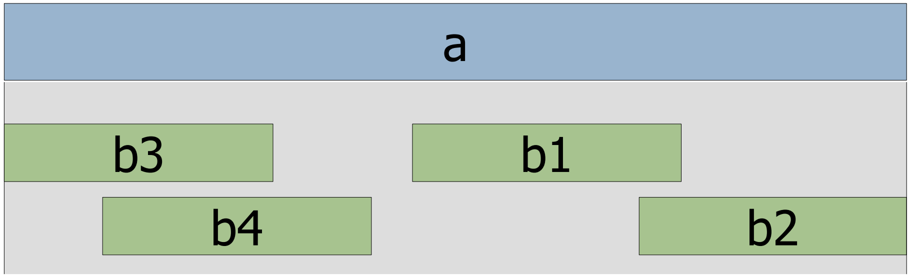


- Properties
  - Complexity is **independent of the time scale**
  - CP Optimizer is able to reason **globally** over a state function
  - **Avoid quadratic models** over each pair (i,j) of incompatible intervals on the state function

Concept: **span constraint**

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

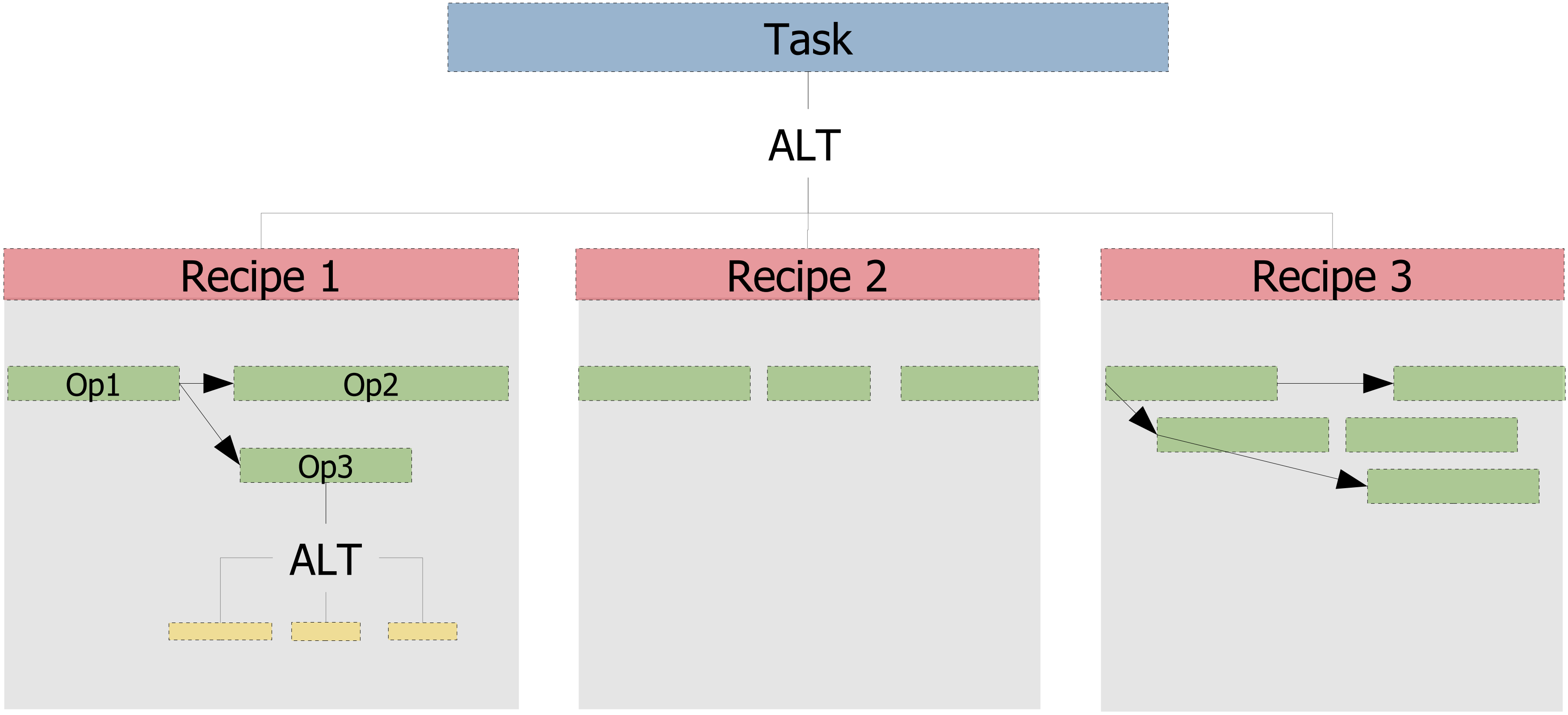
```
span(a,[b1,...,bn])
```



- Properties
  - The constraint of course handles optional interval variables

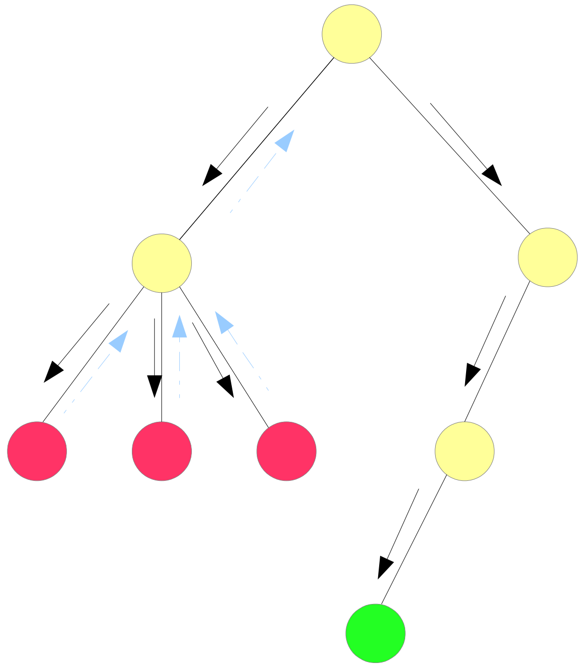
Concept: **span/alternative constraint**

- What for?
  - Modeling Work Breakdown Structure
- Example



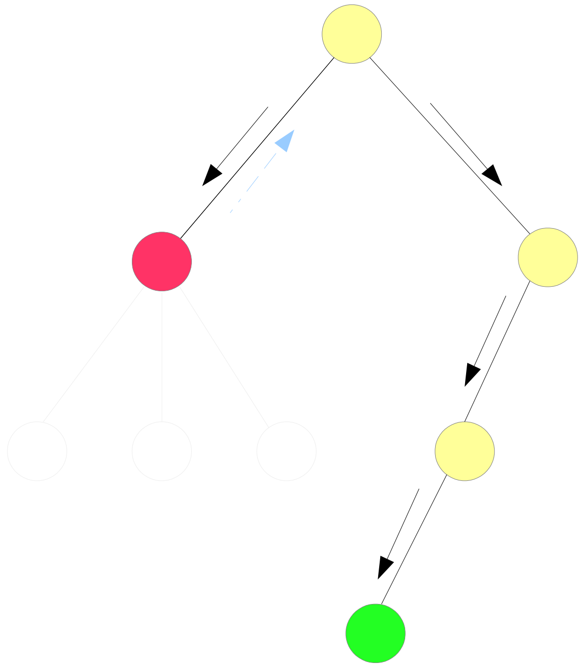
How Constraint Programming Works

- CP is a *constructive* approach
- Values are assigned to variables one at a time to extend a partial solution to a complete solution
- At a point, it may be useless to further extend a partial solution as at least one constraint is already violated by the partial solution
  - The solver *backtracks* and tries a different value for a previously assigned variable
  - All possible assignments of values to variables can be examined in this way



How Constraint Programming Works

- In CP, the basic search behavior is tree search
- Including search space reduction via *domain filtering*
- Domain filtering
  - Before each value-variable assignment, *domain filtering* occurs
  - Each value of a variable which cannot be used in a solution (given the current partial assignment) can be removed
  - Each constraint type has a *specialized* algorithm which filters domains



Domain filtering

All Different

A	B	C	D	E
1	1	1	1	1
2				2
3	3	3		
4		4	4	4
5				5

Domain filtering

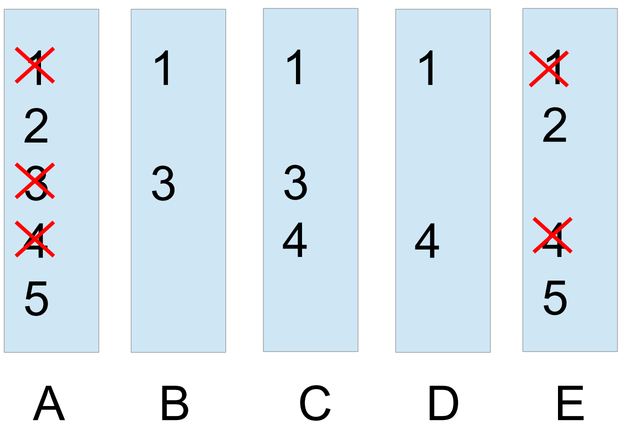
All Different

A	B	C	D	E
~~1~~	1	1	1	~~1~~
2				2
~~3~~	3	3		
~~4~~		4	4	~~4~~
5				5

Uses matching theory

Domain filtering

All Different



Uses matching theory

Other examples in CP Optimizer

- Cardinality (occurrence count)
  - Flow theory
- Packing
  - Subset sum and packing theory
- Table (assignments)
  - Support structures
- Scheduling constraints
  - Energetic reasoning
  - Timetables
  - Edge finding

Model Presolve

- Depending on the input data, models can have redundancies and can contain poorly formulated constraints
- Objective: Automatically improve the model in order to make stronger inferences faster
- Simplifications
  - Constraint compaction
  - Redundancy elimination
  - Constant propagation
  - Common sub-expression factorization
- Aggregations
  - Count expressions & difference constraints
  - Linear constraints over binary {0,1} variables
  - Alternative constraint combined with arithmetic expressions

Examples of Model Presolve

- Elimination of redundant constraints
  - Bound of variables and expressions computed by constraint propagation are used to eliminate redundant constraints before search
- Propagation of constant variables and expressions

$$1 + 2x + 3y + xy - 3x + 7y \leq 8x - 7y + 10$$

$$x == 4 \quad \rightarrow \quad 21y \leq 45$$

- Variable merge

$x==y,\ y==z,\ z==t$ → merge the domains of x, y, z and t and replace y, z, and t by x everywhere

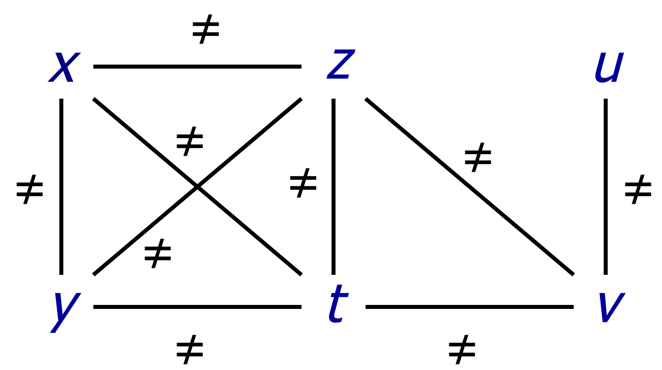
Examples of Model Presolve

- Aggregation of difference constraints
  - Binary ≠ constraints can be lifted to alldiff constraints

A problem with ≠

$$\begin{cases} x \neq y \\ x \neq z \\ x \neq t \\ y \neq z \\ y \neq t \\ z \neq v \\ t \neq z \\ t \neq v \\ u \neq v \end{cases}$$

The associated graph structure



The aggregated model

$$\begin{cases} \texttt{alldiff}(x, y, z, t) \\ \texttt{alldiff}(z, t, v) \\ u \neq v \end{cases}$$

Examples of Model Presolve

- Lifting difference constraints
  1) Pick up a binary constraint $x \neq y$ not already in a lifted alldiff
  2) Find the largest clique containing $x \neq y$ (in practice we use a greedy algorithm – add vertex one by one to the set starting with vertices having the maximum degree)
  3) Add a new lifted alldiff
- Experimentation on graph coloring
  - Color the vertices of a graph such that two adjacent vertices have a different color and the number of colors is minimized
  - The model is stated with a set of binary ≠ constraints
  - Problems are from the DIMACS challenge (COLOR02 set)
    - 9 problems over 60 are solved to optimality without presolve
    - 39 problems over 60 are solved to optimality with presolve

Examples of Model Presolve

Common sub-expression factorization

- Eliminate multiple occurrences of the same expression and replace them by a new variable

  example: $xy \neq z + t$

  $z + xy == a + b$

  $100 \leq z + xy$

- The expressions xy and $z + xy$ appear several times. We introduce two new variables u and v to replace these expressions and add the constraints: $u == xy$, $v == z + u$
- The model becomes $u \neq z + t$

  $v == a + b$

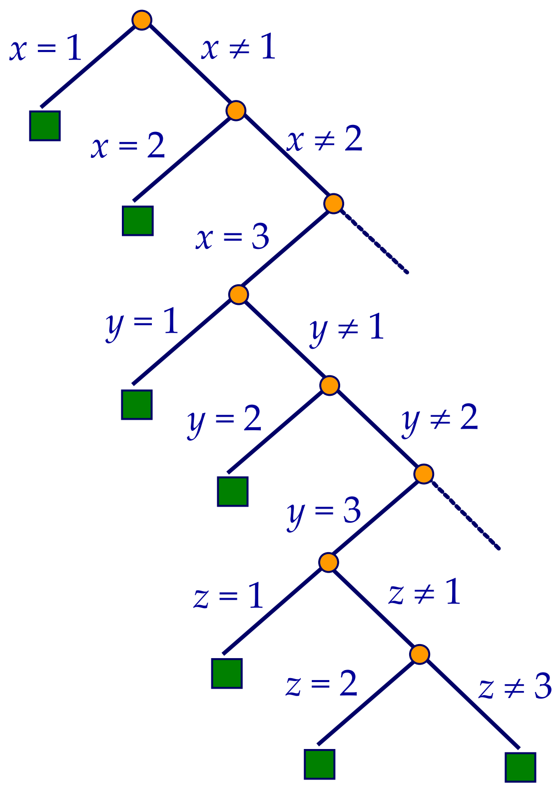
  $100 \leq v$

- Communication of bound reduction on newly introduced variables achieves more domain reduction
- It also reduces the number of expressions and thus involves less computations

Restart and No-goods

- Pure DFS is generally not used
  - Thanks to a parameter, you can select pure DFS if you wish
- Numerous DFS searches of limited effort are favored
  - One element of the CP Optimizer search is the restart
  - Effort limit increases over time to ensure completeness
  - CP Optimizer's restart limit increases geometrically
  - No-good learning helps increase efficiency
  - Benefits well documented in numerous papers:
    - Selman and Gomes, late 90s, theoretical and practical results

Limited DFS Methods - Classic Restarting



Clause Generation

$$\begin{cases} x \neq 1, \\ x \neq 2, \\ x \neq 3 \vee y \neq 1, \\ x \neq 3 \vee y \neq 2, \\ x \neq 3 \vee y \neq 3 \vee z \neq 1, \\ x \neq 3 \vee y \neq 3 \vee z \neq 2, \\ x \neq 3 \vee y \neq 3 \vee z \neq 3, \end{cases}$$

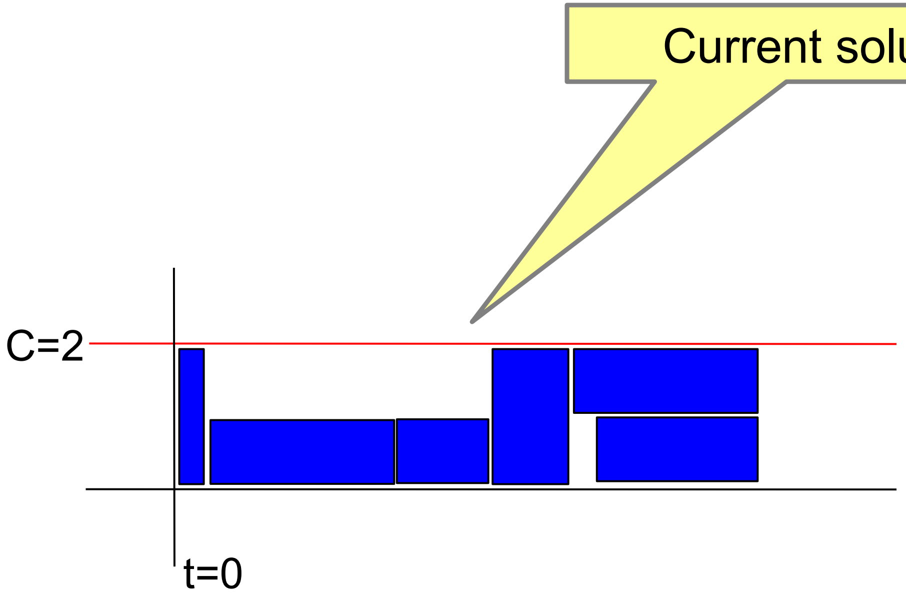
Restart

Increase limit

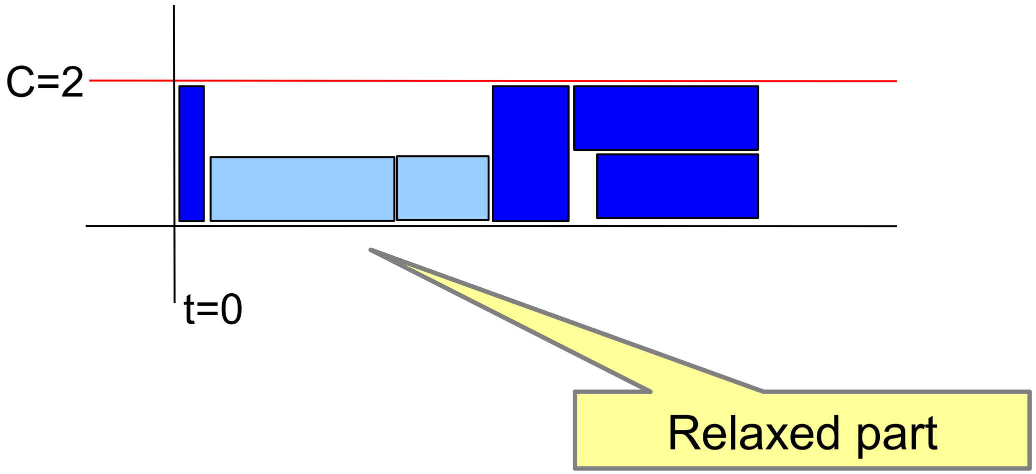
Large Neighborhood Search

- In scheduling, a common way of improving solutions is to use Large Neighborhood Search
  - Shuffling [A&C, 1991] was the first variant
- A number of operations are “relaxed” meaning they can move freely where they want while still obeying problem constraints, such as precedences
- The remaining operations stay “rigid” which means that they can shift in time but stay in much the same order as in the current solution
- The start times of the free and rigid parts are then decided by a limited DFS search using heuristics. The aim is to find a solution of lower objective value

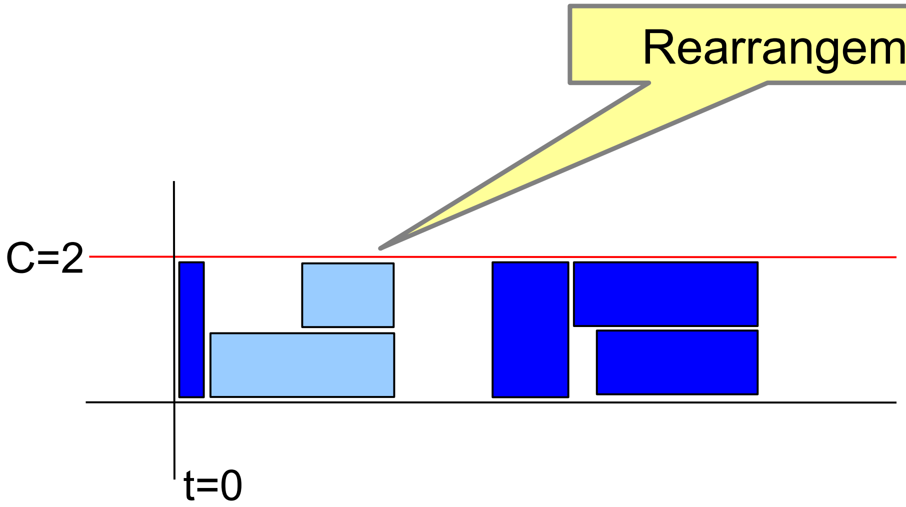
LNS: simple example



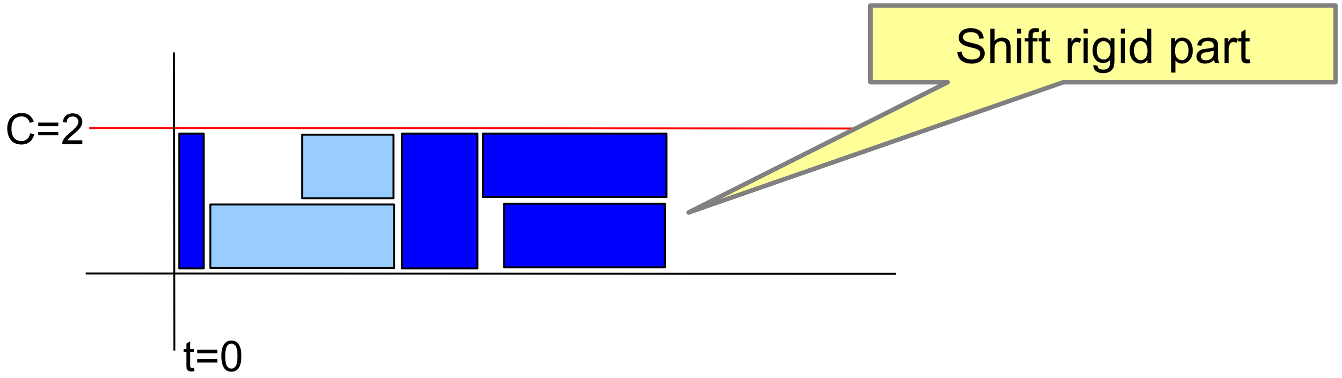
LNS: simple example



LNS: simple example



LNS: simple example

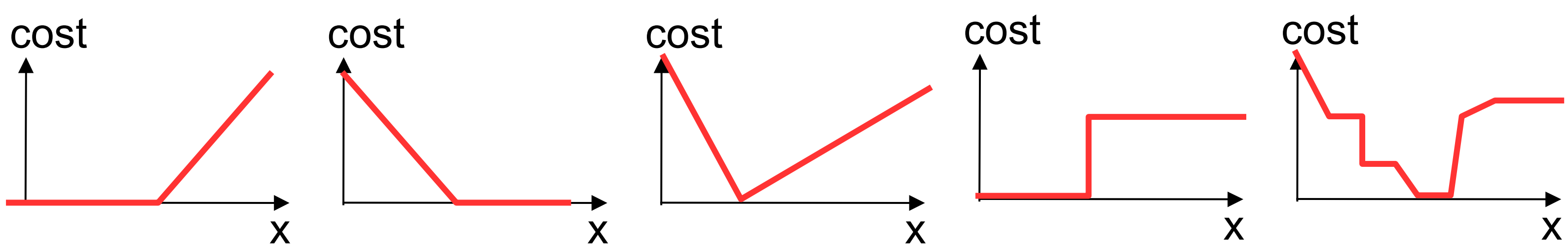


Large Neighborhood Search

- In a classic form of shuffling for job-shop scheduling, the order of operations on one or more resources is fixed and the order of the others is reoptimized
- Problem structure is used to identify the part of the problem to relax and reoptimize. e.g. use resources, alternatives, strongly connected components
- Due to the more general forms of resource supported in CP Optimizer, more subtle forms of "fixing" (creating rigidity in the schedule) are required than simple order
- Use of partial order schedules
  - See [Godard et al., MISTA, 2007], [Pisinger and Ropke, Handbook of Metaheuristics, 2010]

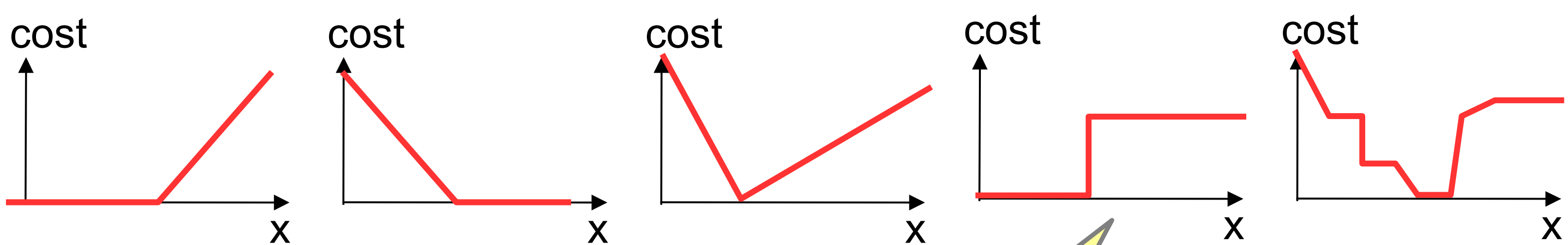
Branching Heuristics – LP-assisted

- Traditionally, early/tardy scheduling problems are challenging for CP solvers as the solvers don't have a good global view of the cost function



Branching Heuristics – LP-assisted

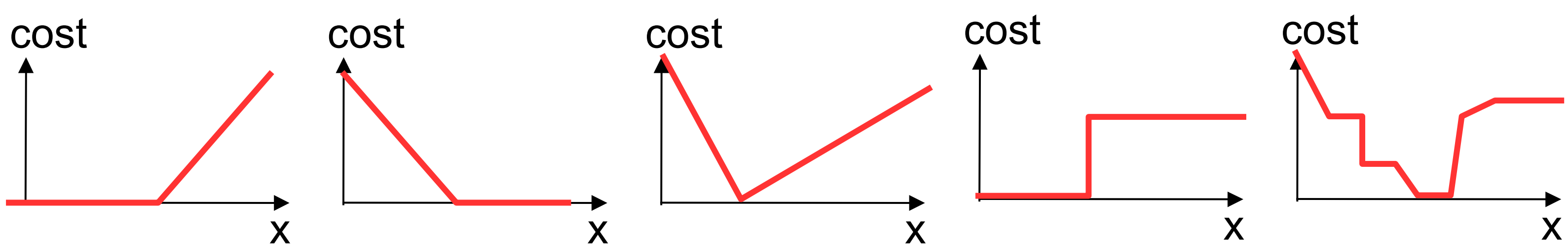
- Traditionally, early/tardy scheduling problems are challenging for CP solvers as the solvers don't have a good global view of the cost function



penalty * (startOf(op) > dueDate)

Branching Heuristics – LP-assisted

- Traditionally, early/tardy scheduling problems are challenging for CP solvers as the solvers don't have a good global view of the cost function

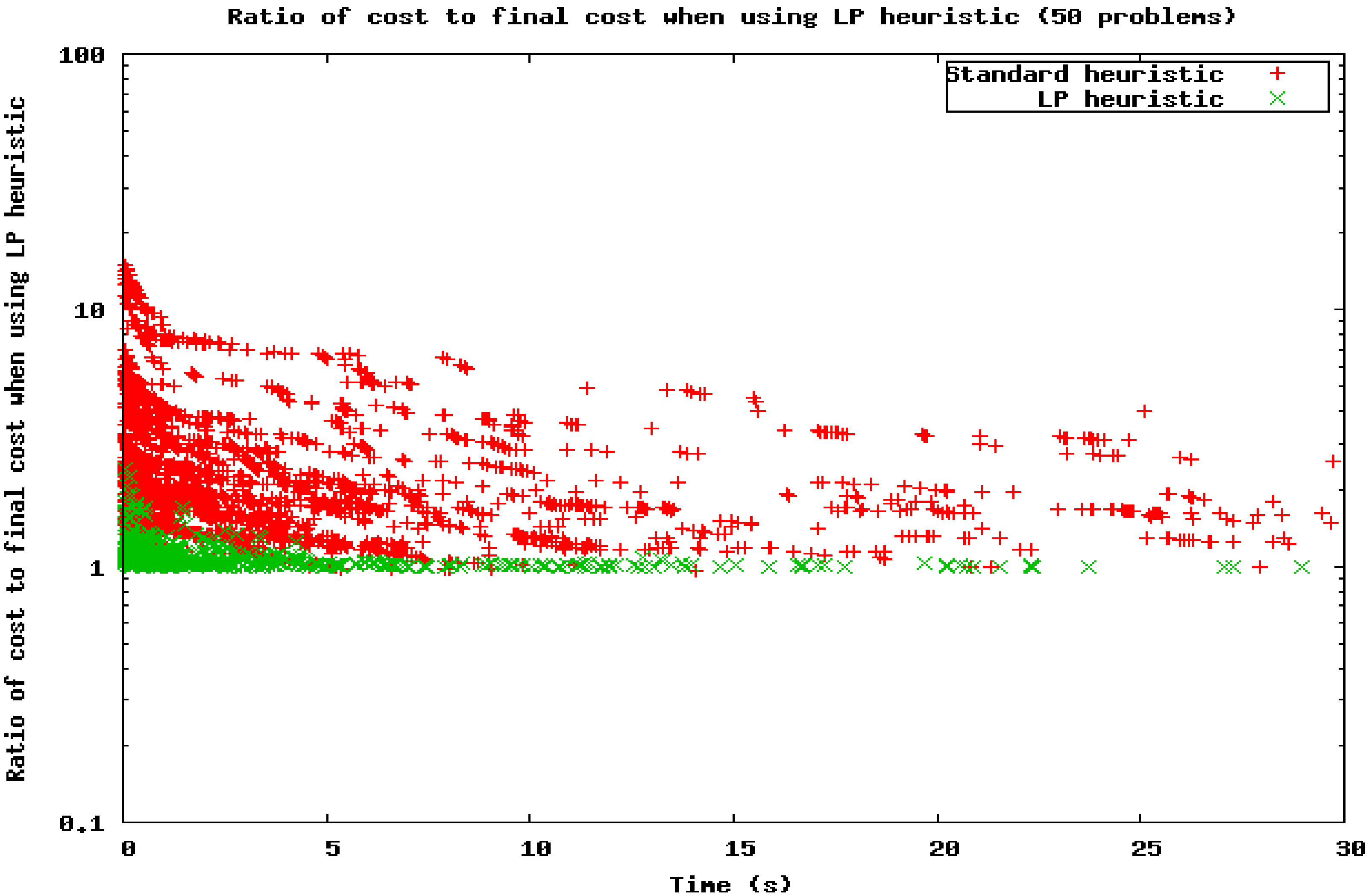


max(0, slope * (startOf(op) - dueDate))

Branching Heuristics – LP-assisted

- Since CPLEX and CP Optimizer are both part of CPLEX Optimization studio, CP Optimizer can take advantage of the availability of CPLEX to help on problems where a good global view of the cost function is important
- CP Optimizer linearizes scheduling constructs such as:
  - Precedences
  - Execution / non-execution costs, alternatives
  - Cost function terms which are functions of start/end times
  - PWL functions can be used to approximate
- CPLEX computes cost lower bound and “best” start times using the CPLEX LP Solver: use the LP solution to guide heuristics. Start an operation as close as possible to the time proposed by CPLEX

Branching Heuristics – LP-assisted



Probing

- Probing is a technique that exploits the fact that in some problems, fixing a variable has a higher than usual probability of causing a failure
  - In this case, at a node other variables as well as the branching variable can be tested by fixing them to a value
  - If the fixing fails, the fixed value can be filtered from the domain, otherwise the fixing is undone
  - Particularly useful for binary and small-domain variables
- In CP Optimizer, a dynamic technique is used to adjust how often probing occurs to avoid paying the probe penalty when probing is ineffective

IBM®

Legal Disclaimer