An introduction to CP Optimizer

Philippe Laborie
IBM Analytics, Decision Optimization

May 25, 2016

Outline

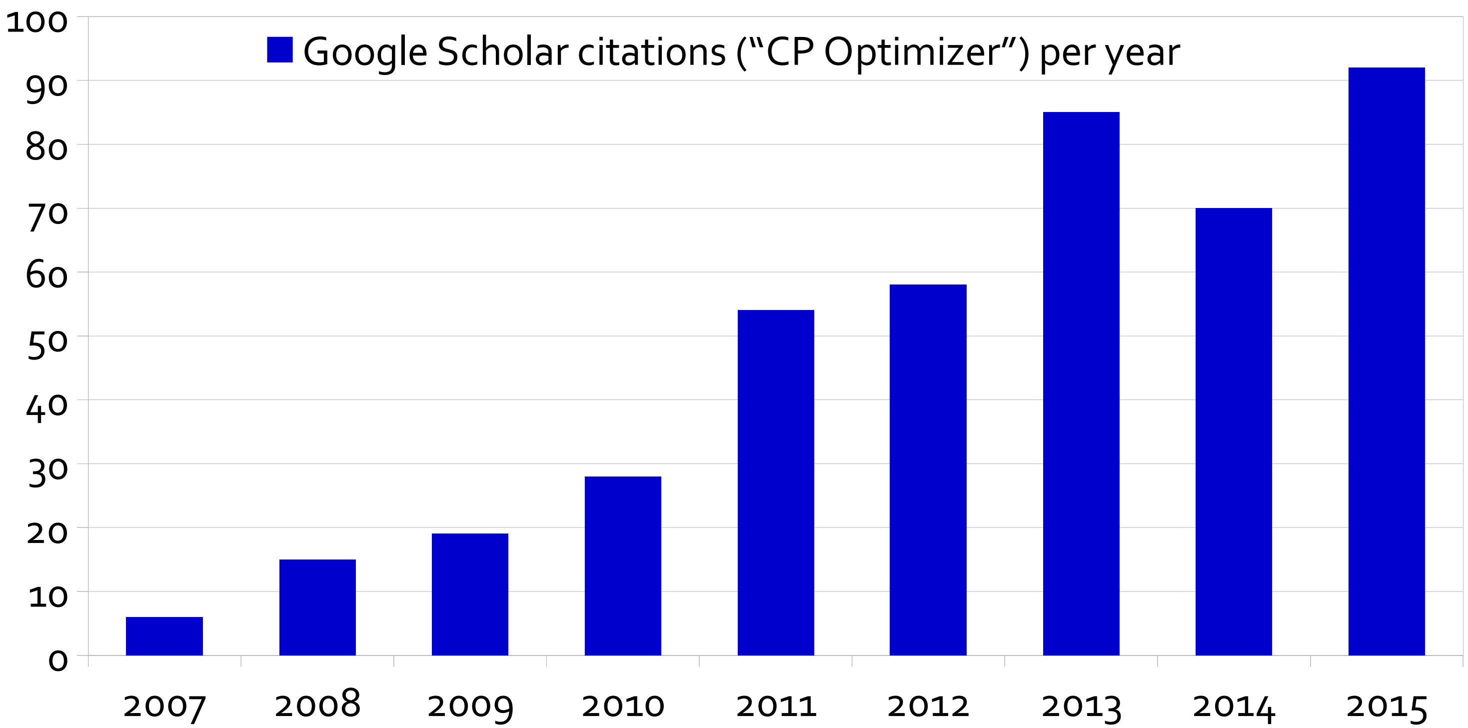
1) Overview of CP Optimizer
2) Modeling concepts
3) Automatic search
4) Development tools

Overview of CP Optimizer

- A component of **IBM ILOG CPLEX Optimization Studio**
- A **Constraint Programming** engine for combinatorial problems (including **scheduling problems**)
- Implements a **Model & Run** paradigm (like CPLEX)
  - Model: **Concise** yet **expressive** modeling language
  - Run: **Powerful automatic search procedure**
    Search algorithm is **complete**
- Available through the following interfaces:
  - OPL
  - C++ (native interface)
  - Python, Java, .NET (wrapping of the C++ engine)
- Set of **tools** to support the development of efficient models

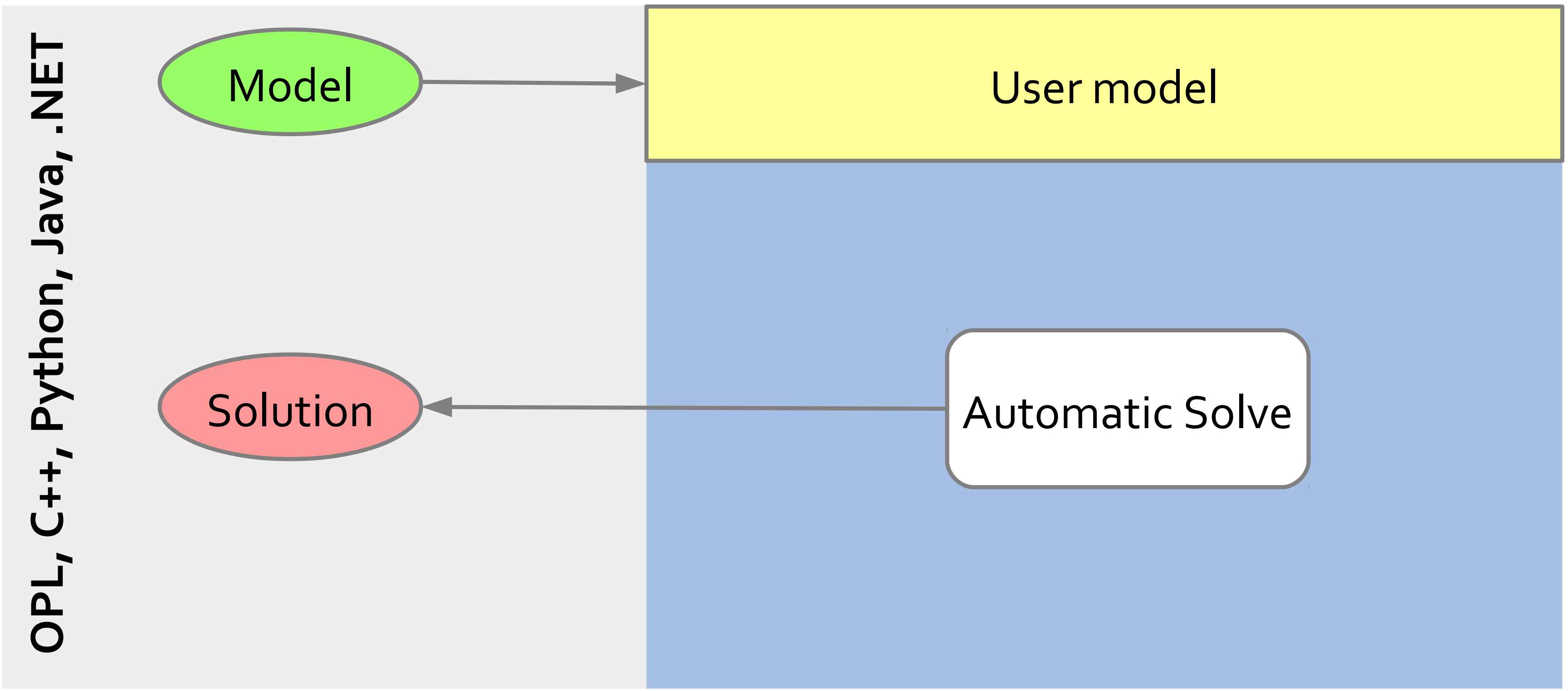
Overview of CP Optimizer

- First version in 2007/2008
- Search “CP Optimizer” on Google Scholar to get an idea of how CP Optimizer is being used across academy and industry

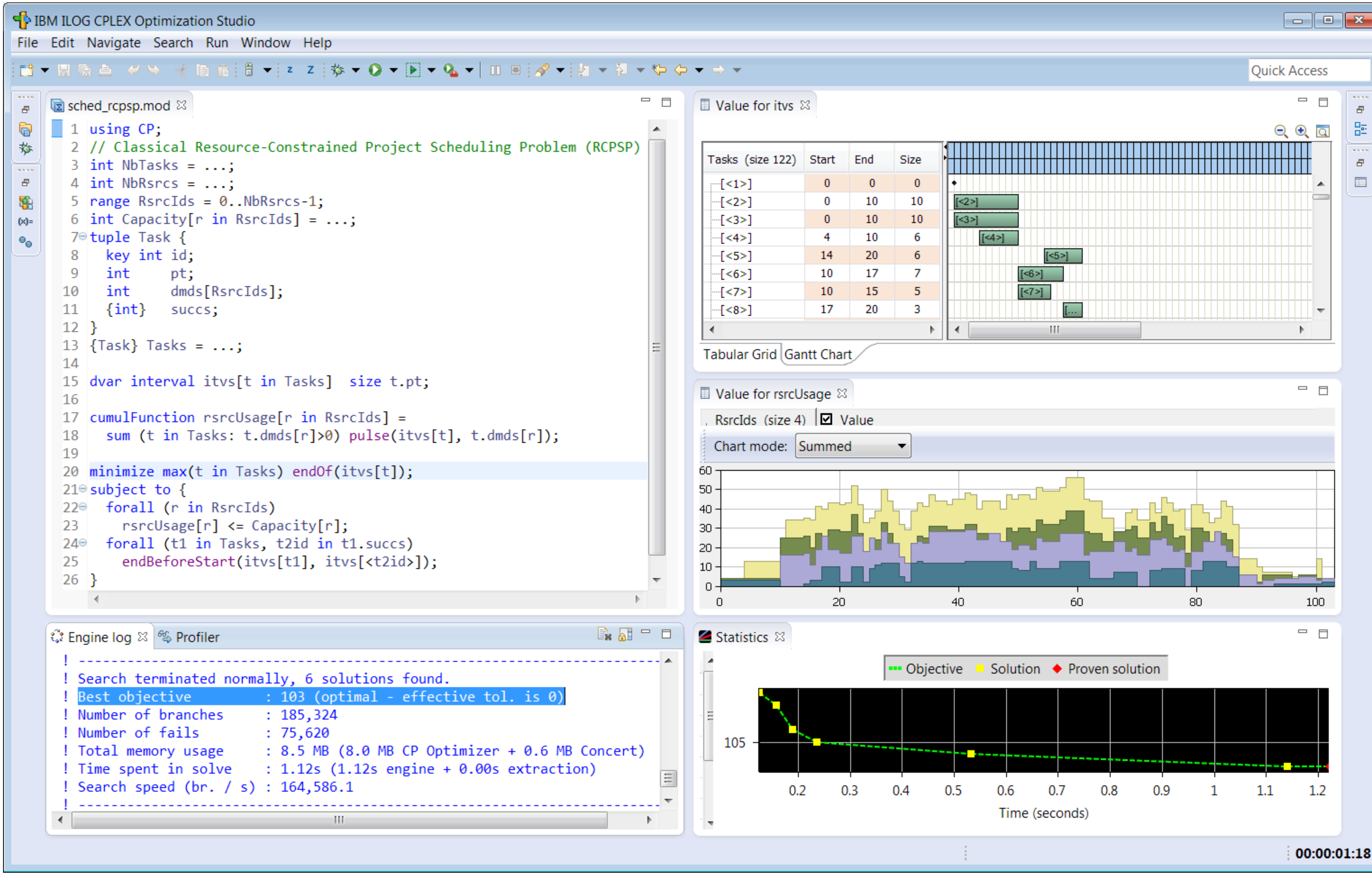


Overview of CP Optimizer

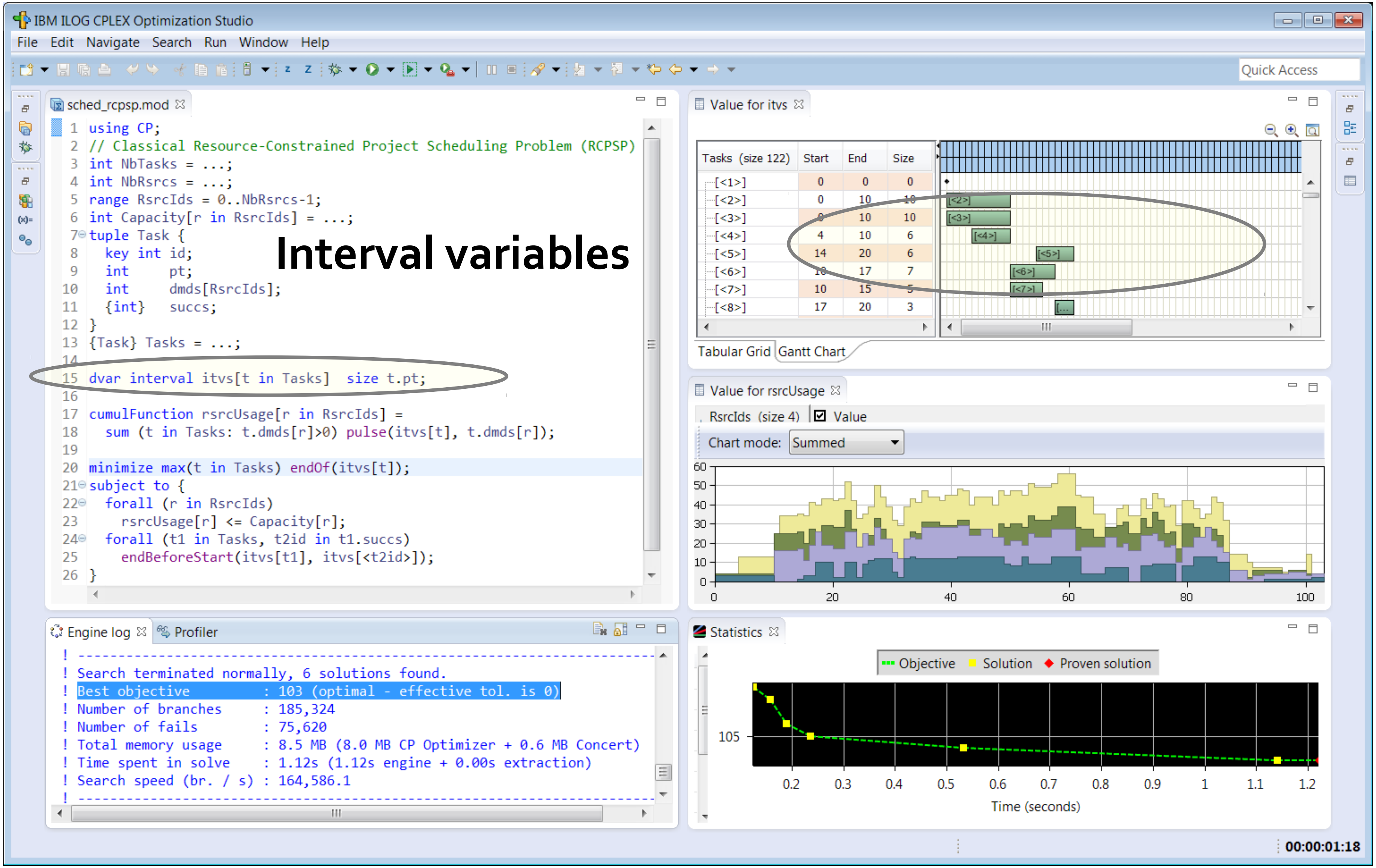
- Model & Run paradigm



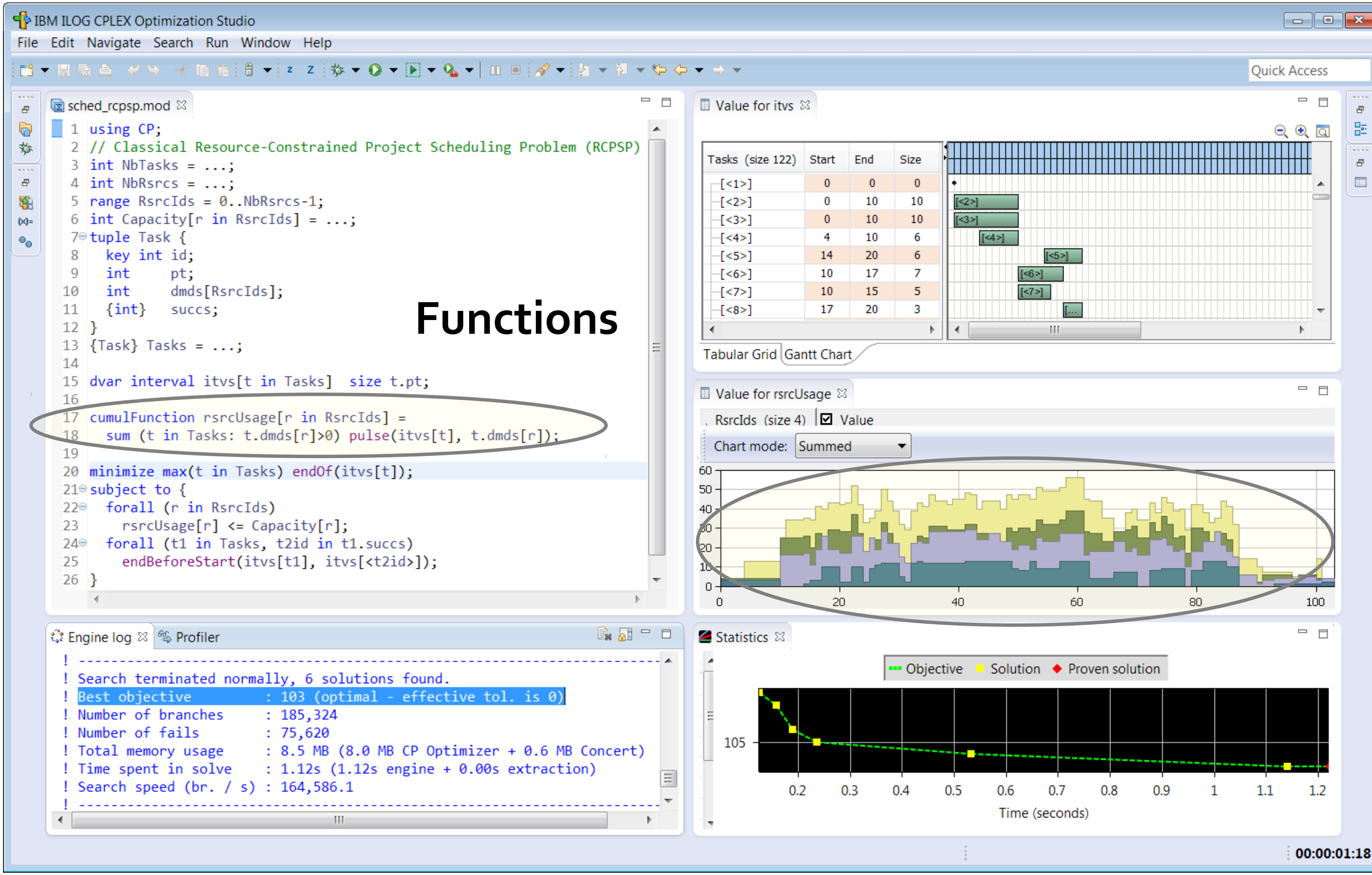
Overview of CP Optimizer



Overview of CP Optimizer



Overview of CP Optimizer



Modeling concepts

- Two main types of decision variables:
  - Integer variables
  - Interval variables

Modeling concepts (integer variables)

<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. x = (y < z)

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

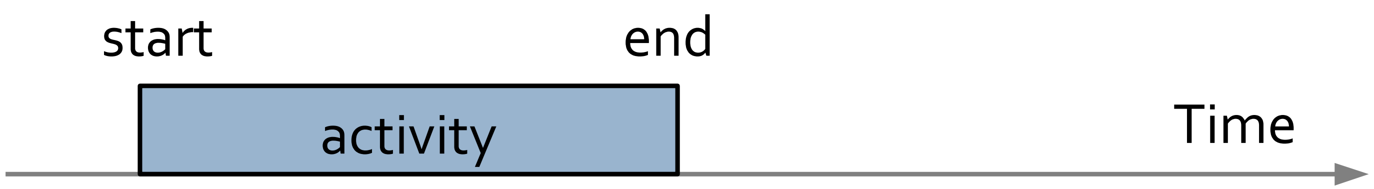
Standard relational constraints (==, !=, <, >, <=, >=)

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

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

Modeling concepts (interval variables for scheduling)

- Scheduling (our definition of):
  - 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 …)

Modeling concepts (interval variables for scheduling)

- Extension of classical CSP with a new type of decision variable: **optional interval variable** :

$$\text{Domain}(x) \subseteq \{\bot\} \cup \{\, [s,e) \mid s,e \in \mathbb{Z},\ s \le e \,\}$$

Absent interval

Interval of integers

- Introduction of mathematical notions such as **sequences** and **functions** to capture temporal aspects of scheduling problems

Modeling concepts (interval variables for scheduling)

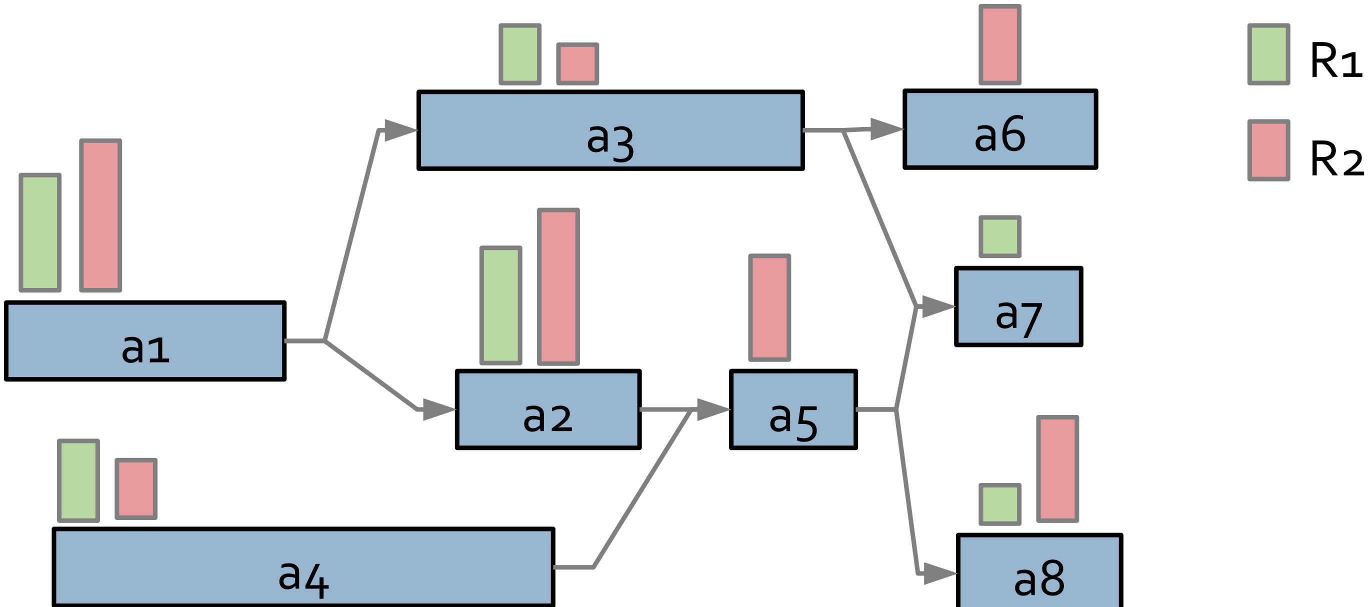
- In scheduling models, interval variables usually represent an interval of time whose end-points (start/end) are decision variables of the problem
- Examples:
  - A production order, an operation in a production order
  - A sub-project in a project, a task in a sub-project
  - A batch of operations
  - The setup of a tool on a machine
  - The moving of an item by a transportation device
  - The utilization interval of a machine
  - The filling or emptying of a tank
- Idea of the model (and search) is to avoid the enumeration of start/end values

Modeling concepts (interval variables for scheduling)

- An interval variable can be **optional** meaning that it is a decision to have it present or absent in a solution.
- Examples:
  - Unperformed tasks and optional sub-projects
  - Alternative resources, modes or recipes for processing an order, each mode specifying a particular combination of operational resources
  - Operations that can be processed in different temporal modes (e.g. series or parallel), left unperformed or externalized
  - Activities that can be performed in an alternative set of batches or shifts

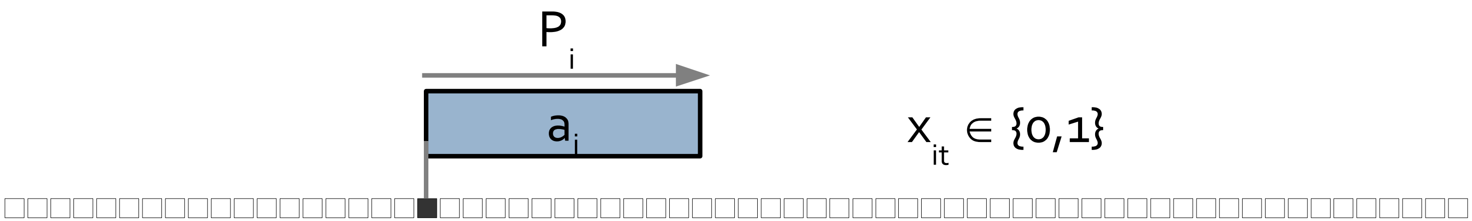
Example: Resource Constrained Project Scheduling Problem

- 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

- Basic CP Optimizer model for RCPSP:

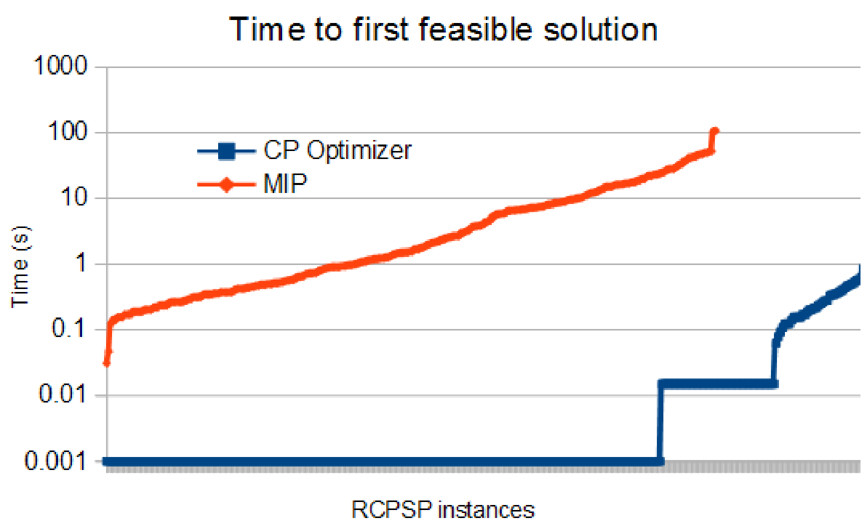
```
dvar interval a[i in Tasks] size i.pt;

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>]);
}
```

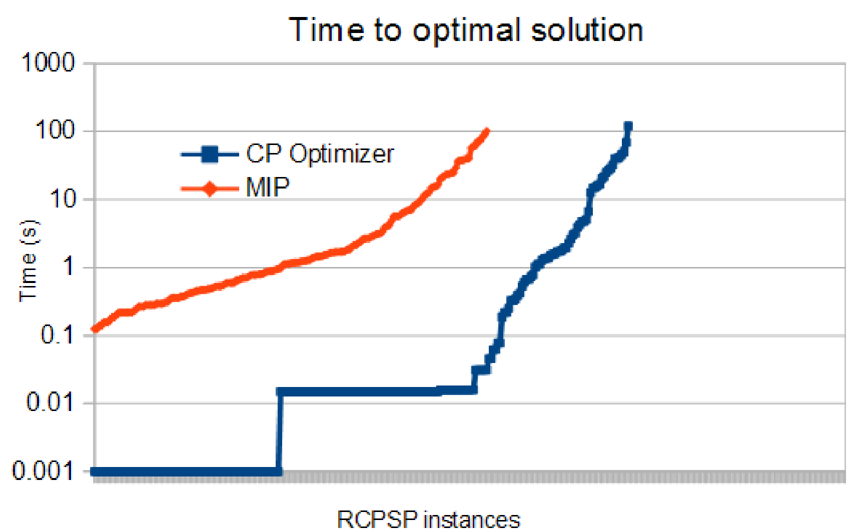
Example: Resource Constrained Project Scheduling Problem

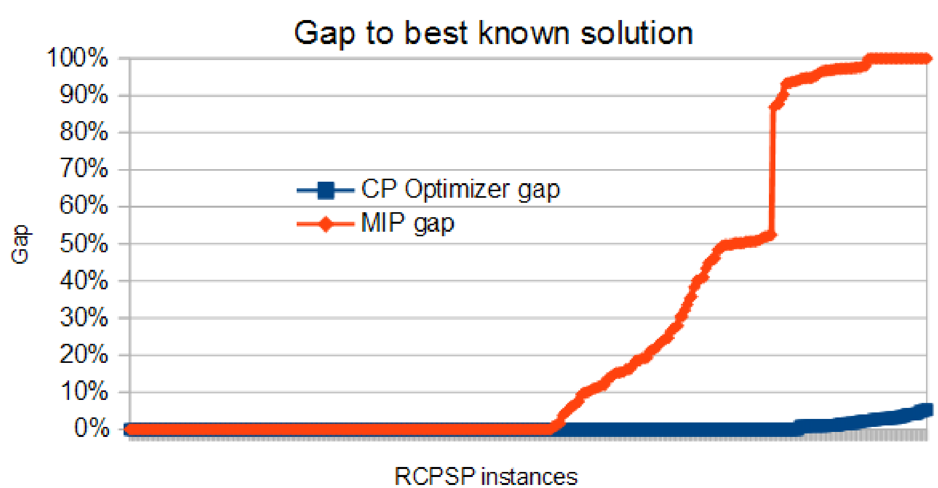
- Comparison of this time-indexed MIP formulation against the CP Optimizer model on a set of:
  - 300 classical **small** RCPSP instances (30-120 tasks) +
  - 40 slightly **more realistic** larger ones (900 tasks)
  - time-limit: 2mn, 4 threads
- Note: industrial scheduling problems are often much **larger**, typically several 1.000 tasks (we handled up to 1.000.000 tasks in an RCPSP-like scheduling application in V12.6)

Example: Resource Constrained Project Scheduling Problem

- Comparison of CP Optimizer and MIP performance on RCPSP

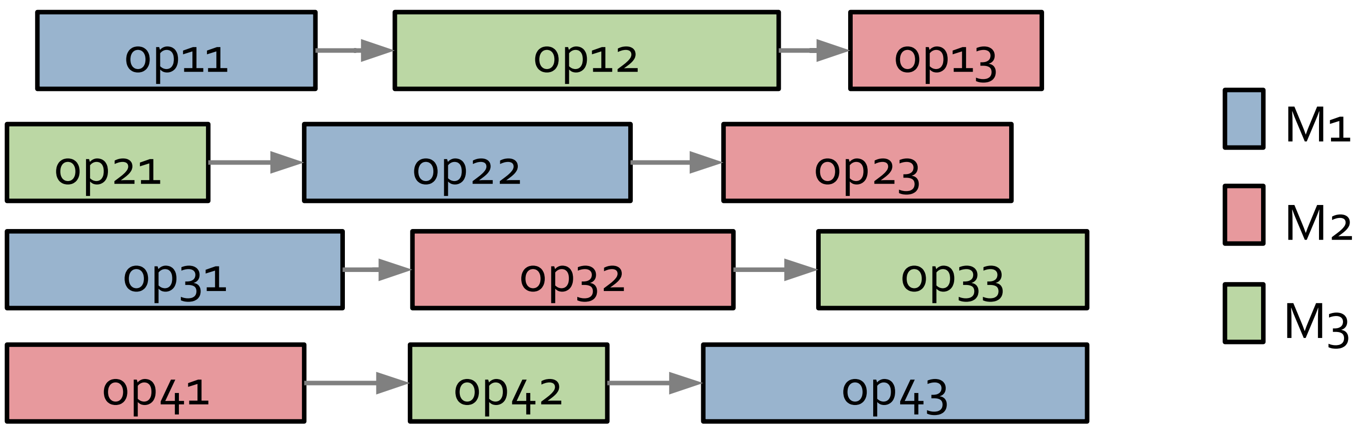






Example: Job-shop Scheduling Problem

- Example: Job-shop Scheduling Problem



- Minimization of makespan

Example: Job-shop Scheduling Problem

- 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]);
}
```

Example: Job-shop Scheduling Problem

- 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:

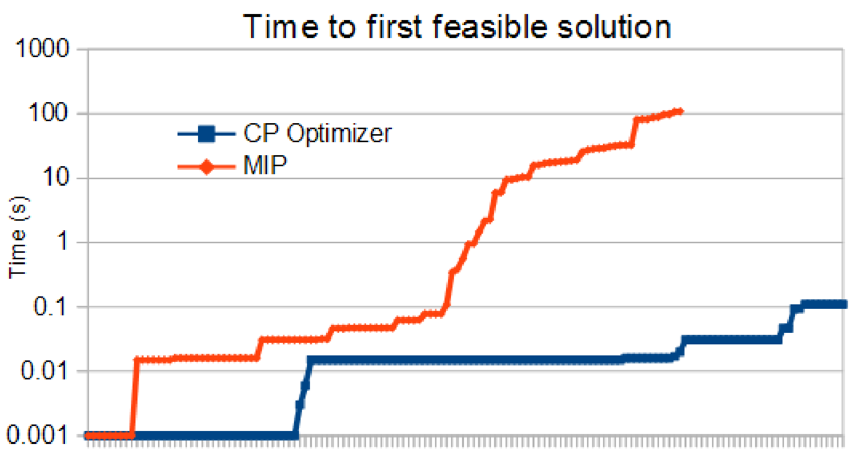
    noOverlap(all(i in ...) a[i])

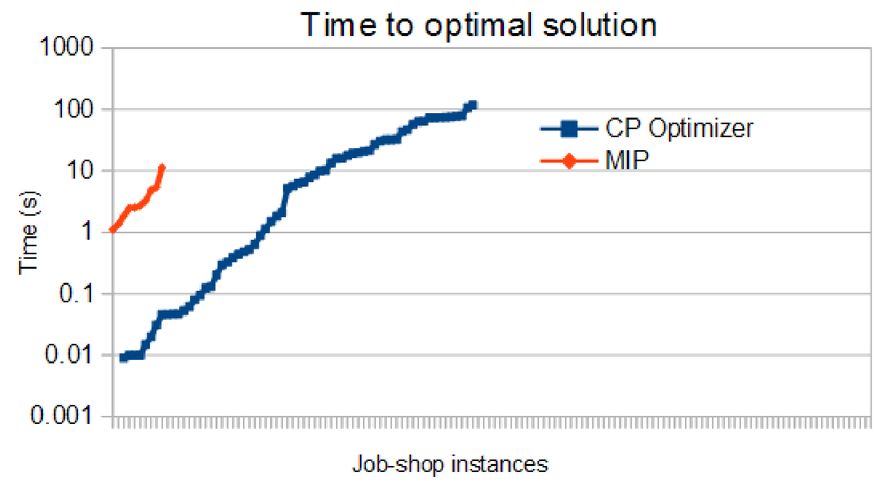
Example: Job-shop Scheduling Problem

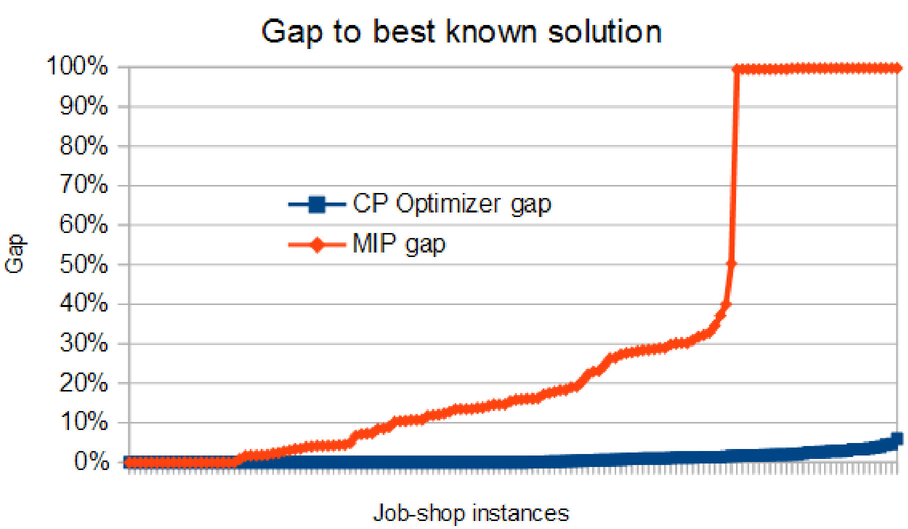
- Comparison of the CP Optimizer model vs a disjunctive MIP formulation on a set of 140 classical Job-shop instances (50-2000 tasks), time-limit: 2mn, 4 threads

Example: Job-shop Scheduling Problem

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







Example: Flexible Job-shop Scheduling Problem

- CP Optimizer model

```
dvar interval ops  [Ops];
dvar interval modes[md in Modes] optional size md.pt;
dvar sequence mchs [m in Mchs] in
  all(md in Modes: md.mch == m) modes[md];

minimize max(o in Ops) endOf(ops[o]);
subject to {
  forall (j in Jobs, o1,o2 in JobOps[j]: o2.pos==1+o1.pos)
    endBeforeStart(ops[o1],ops[o2]);
  forall (o in Ops)
    alternative(ops[o], all(md in Modes: md.opId==o.id) modes[md]);
  forall (m in Mchs)
    noOverlap(mchs[m]);
}
```

CP Optimizer modeling concepts

- CP Optimizer has mathematical concepts that naturally map to features invariably found in industrial scheduling problems

Optional activities
Over-constrained problems
Resource alloc. / alternatives
Work-breakdown structures
— **Optional interval variables**

Cumul functions —
Cumulative resources
Inventories
Reservoirs

State functions —
Parallel batches,
Activity incompatibilities

Sequence variables —
Unary resources
Setup times/costs
Travel times/costs

Calendars
Resource efficiency
— **Intensity functions**

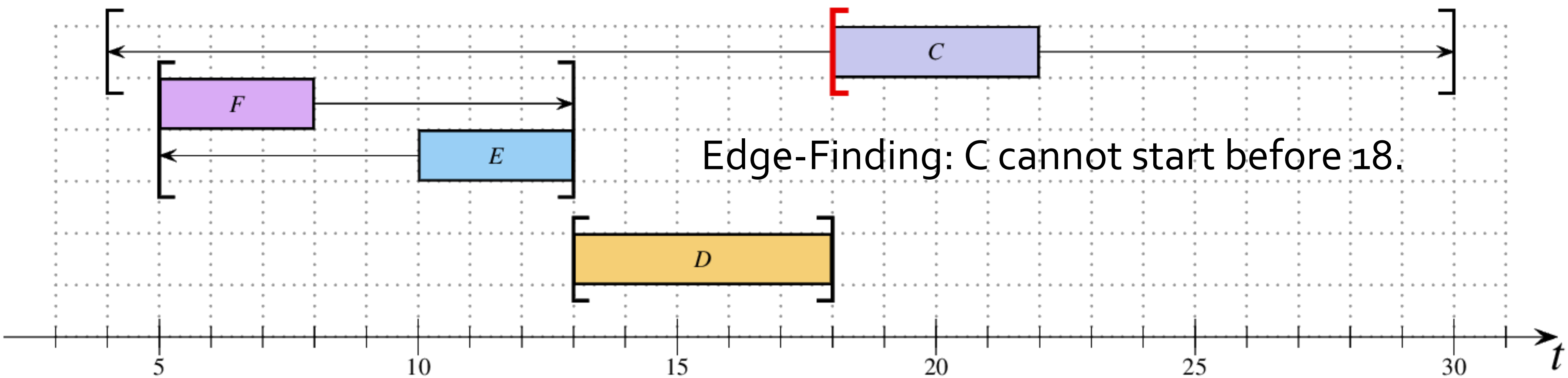
Earliness/tardiness costs
Net Present Value
— **General arithmetical expressions**

Automatic Search

- Search algorithm is **Complete**
- Core CP techniques used as a building block:
  - Tree search (Depth First)
  - Constraint propagation
- But also:
  - Deterministic multicore parallelism
  - Model presolve
  - Algorithms portfolios
  - Machine learning
  - Restarting techniques
  - Large Neighborhood Search
  - No-good learning
  - Impact-based branching
  - Opportunistic probing
  - Dominance rules
  - LP-assisted heuristics
  - Randomization
  - Evolutionary algorithms

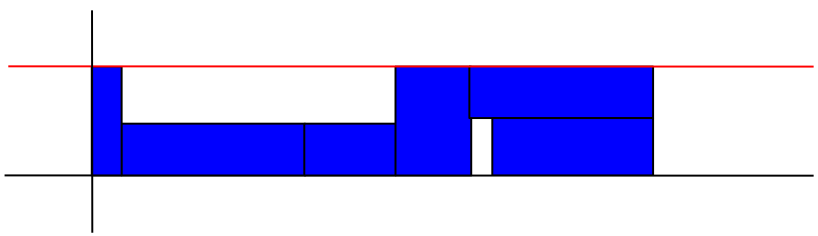
Automatic Search – Constraint propagation

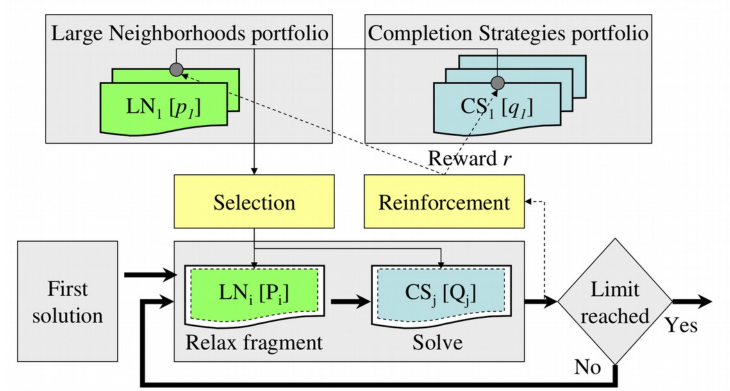
- Dedicated Propagation Algorithms for Scheduling
  - Edge-Finding
  - Not-First/Not-Last
  - Detectable Precedences
  - Timetable
  - Timetable Edge-Finding
  - Max Energy Filtering
  - etc.



Automatic Search – Large Neighborhood Search (LNS)

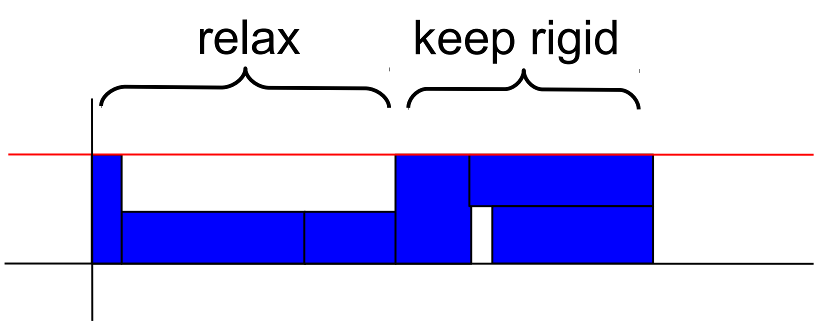
- LNS is able to converge very quickly to quality solutions
  1) Start with an existing solution

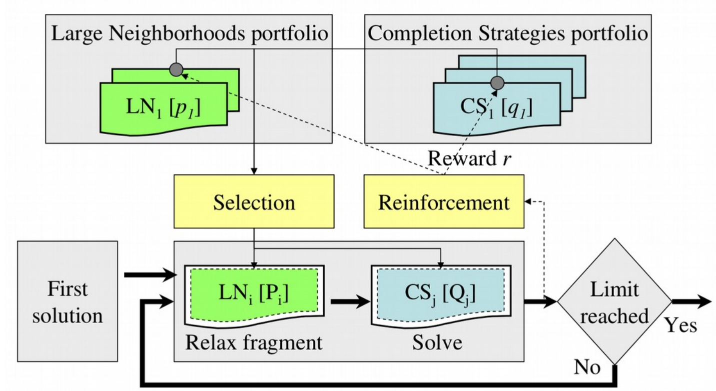




Automatic Search – Large Neighborhood Search (LNS)

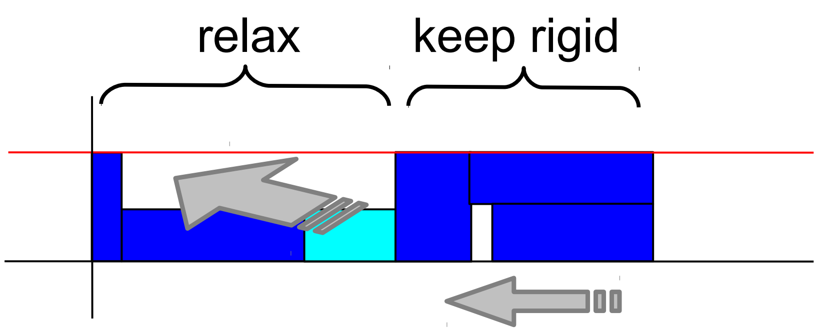
- LNS is able to converge very quickly to quality solutions
  1) Start with an existing solution
  2) Take part the solution and relax it, fix structure of the rest (but not start/end times)

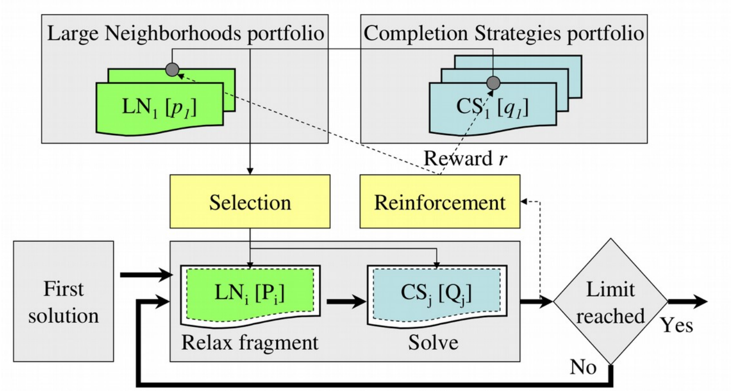




Automatic Search – Large Neighborhood Search (LNS)

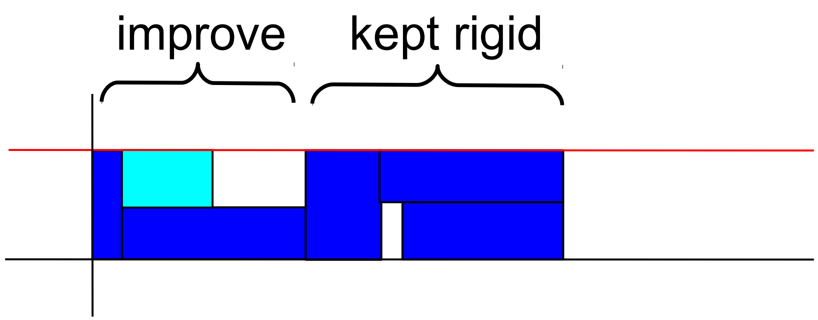
- LNS is able to converge very quickly to quality solutions
  1) Start with an existing solution
  2) Take part the solution and relax it, fix structure of the rest (but not start/end times)

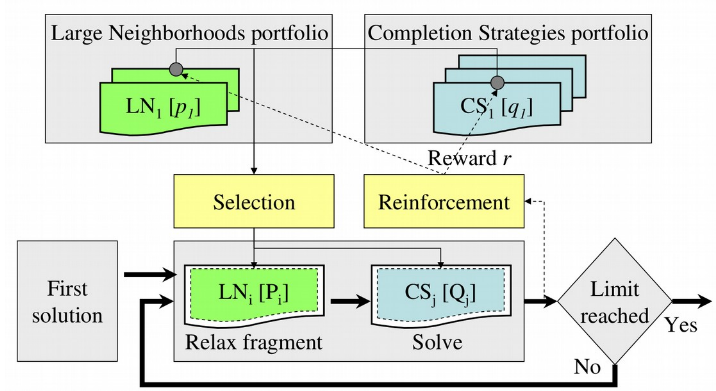




Automatic Search – Large Neighborhood Search (LNS)

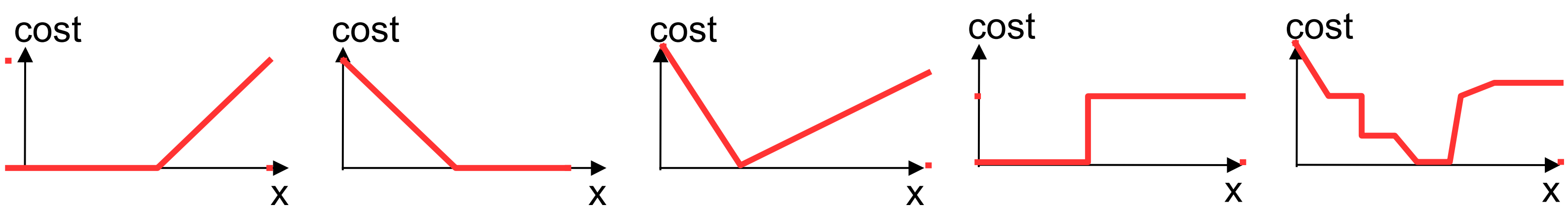
- LNS is able to converge very quickly to quality solutions
  1) Start with an existing solution
  2) Take part the solution and relax it, fix structure of the rest (but not start/end times)
  3) Find (improved) solution





Automatic Search – LP-assisted heuristics

- Traditionally, early/tardy problems are challenging for CP-based scheduling tools as they miss a good global view of the cost



- Approach:
  - Automatically use CPLEX's LP solver to provide a solution to a relaxed version of the problem
  - Use the LP solution to guide heuristics. Start an operation as close as possible to the time proposed by CPLEX

- What is linearized?
  - Precedences
  - Execution / non-execution costs, logical constraints on optional intervals
  - “Alternative” and “Span” constraints
  - Cost function terms which are functions of start/end times

Automatic Search: some results

MISTA 2007

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: some results

CP-AI-OR 2009

- 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: some results

CP-AI-OR 2015

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

Automatic Search: some results

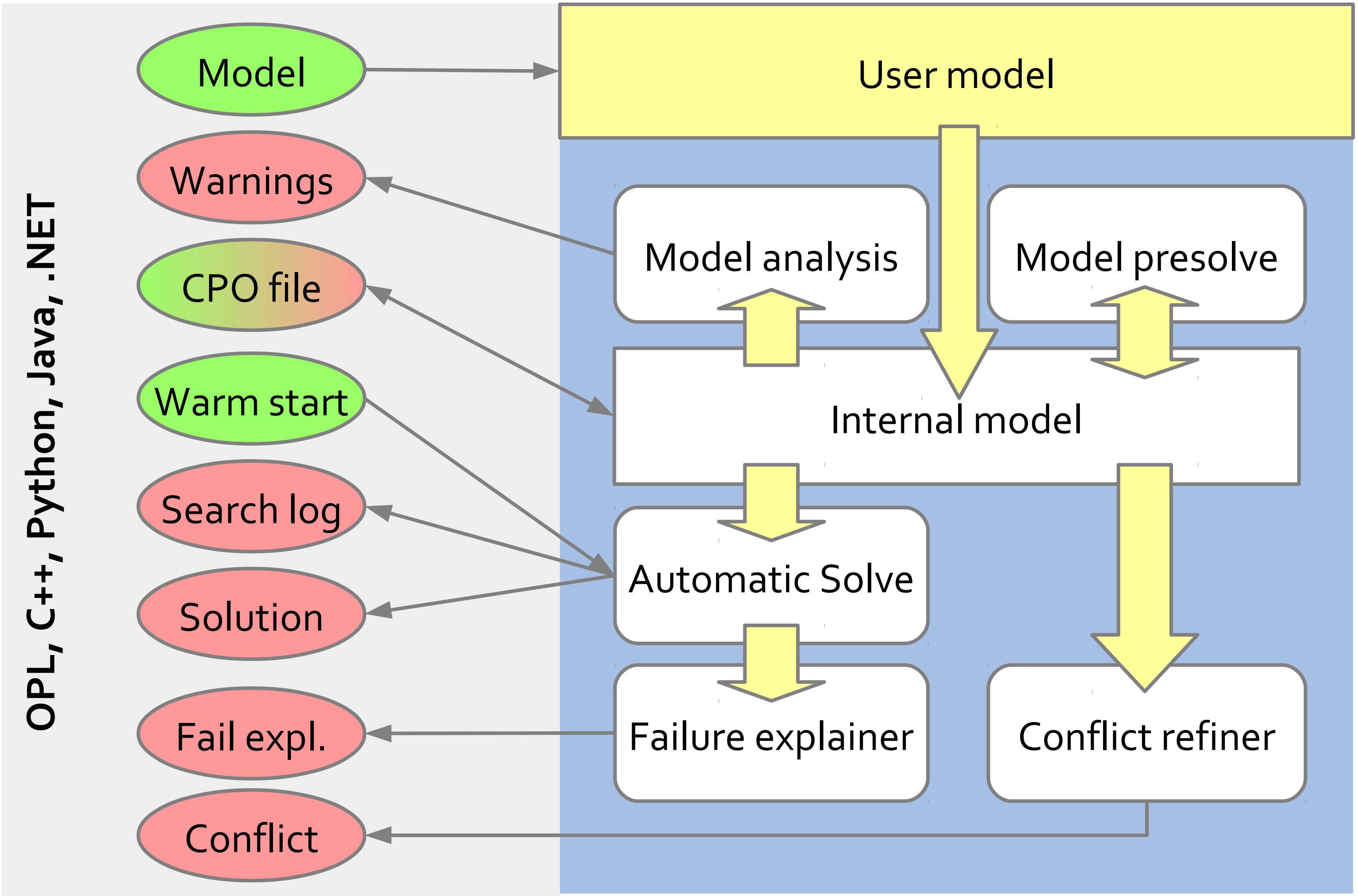
CP 2015

- Industrial Modelling Competition at CP 2015
- http://booleconferences.ucc.ie/indmodellingcomp
- CP Optimizer outperformed all the other competitors on all the instances of the challenge

Automatic Search

- Some CP Optimizer industrial applications:
  - Port Management: Yantian International Container Terminals, Navis
  - Manufacturing: Ajover S.A. (plastics), TAL Group (textiles)
  - Aviation: Dassault Aviation, another large jet manufacturer
  - Workforce scheduling: A world leading IFS (Integrated Facility Services) company

Development tools



CPLEX and CP Optimizer are in the same ecosystem

		CPLEX	CP Optimizer
Interfaces		OPL, C++, Java, .NET, C, Python	OPL, C++, Java, .NET, Python
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	✓	✓