Planning/Scheduling with CP Optimizer

Philippe Laborie
IBM, IBM Data & AI
laborie@fr.ibm.com

Planning/Scheduling with CP Optimizer

- Overview of CP Optimizer
- Typical applications
- Modeling concepts
- Automatic search
- Performance
- Tools

Overview of CP Optimizer

- Developed by ILOG/IBM since 2007
- Focus on industrial / real life optimization problems
- Targeted audience goes beyond CP experts:
  - OR experts
  - Data scientists
  - Software engineers
- The CP Optimizer approach: **Model** & **run**
  - Declarative mathematical model
    - Introduction of adequate mathematical concepts for formulation of scheduling problems (optional intervals, functions, permutations)
  - No need to worry about the resolution
    - Exact algorithm using hybrid methods
    - Good out of the box performance for real world problems

Overview of CP Optimizer

As a result, CP Optimizer is quite an **atypical** CP solver:

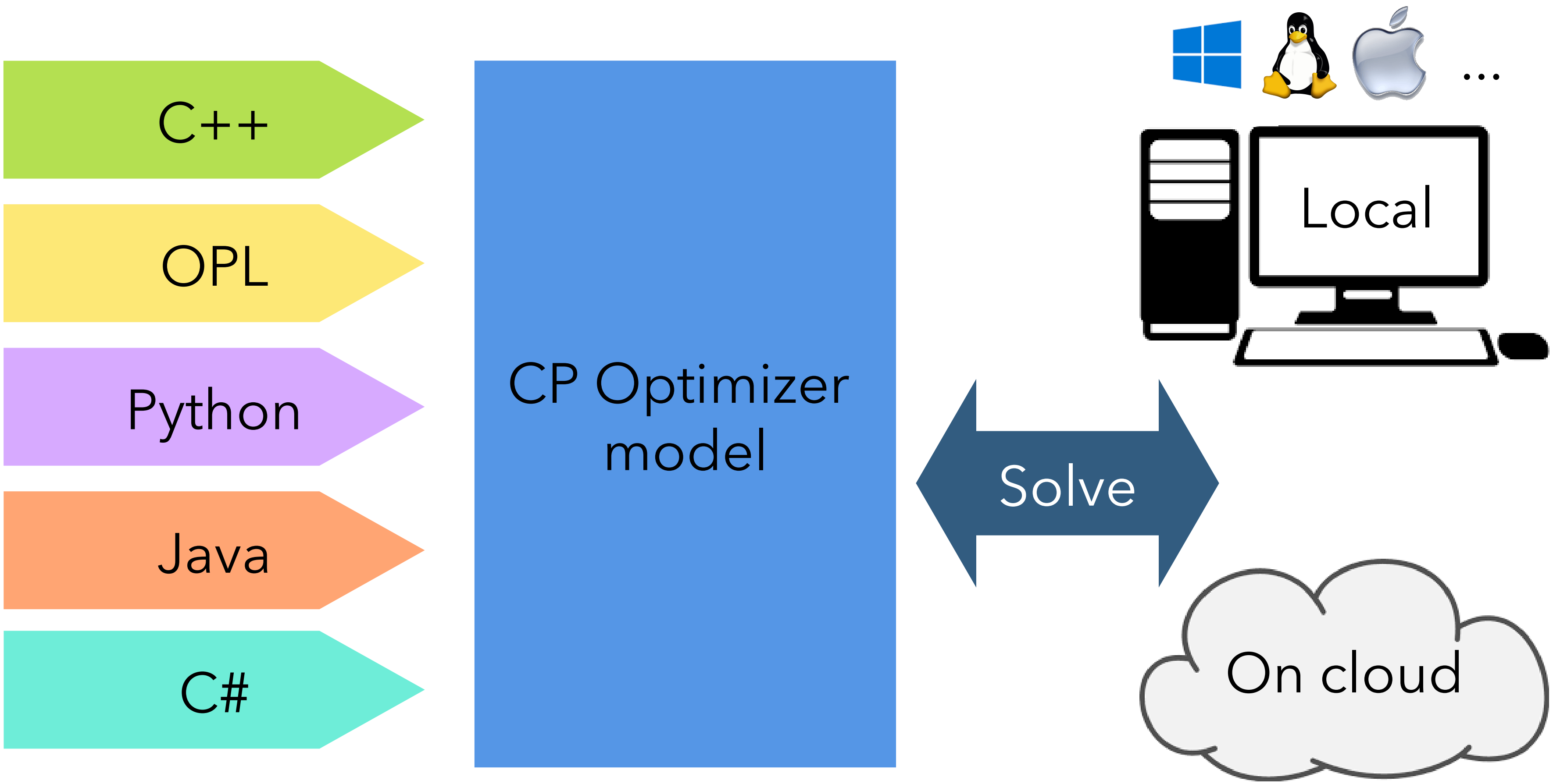
- Focus on Model&Run:
  - Search extension (user-defined constraints and search) is **not encouraged** (though it is possible of course)
  - **Few** search parameters
- **Few** types of constraints / global constraints
- **More** types of variables and expressions due to the introduction of *intervals*, *functions*, *sequences*

Overview of CP Optimizer

As a result, CP Optimizer is quite an **atypical** CP solver:

- Big focus on **optimization** problems (vs feasibility)
- Many other ingredients than "plain" CP in the automatic search (e.g. linear relaxation)
  - CP Optimizer is not "just" an exact algorithm … it also transparently embeds, under the hood, a lot of meta-heuristic search
  - We view computing *good solutions* and computing *bounds* as two (almost) **separate questions** to be addressed by different techniques

Overview of CP Optimizer



A **counter-example** of industrial scheduling problem

The classical **job-shop scheduling** problem

- Resource/machines are over-simplified
  - In reality: setup-times, activities incompatibilities, batching, cumulative resources, inventories (reservoirs), execution conditions (e.g. resource safety levels),...
- All operations are performed in a unique way
  - In reality: resource allocation, optional operations, alternative recipes, hierarchical decomposition
- The makespan objective function is completely unrealistic
  - In reality: combination of earliness/tardiness costs, non-execution cost, resource related costs, constraint violation, …
- Real problems are often much larger than the size of current benchmarks

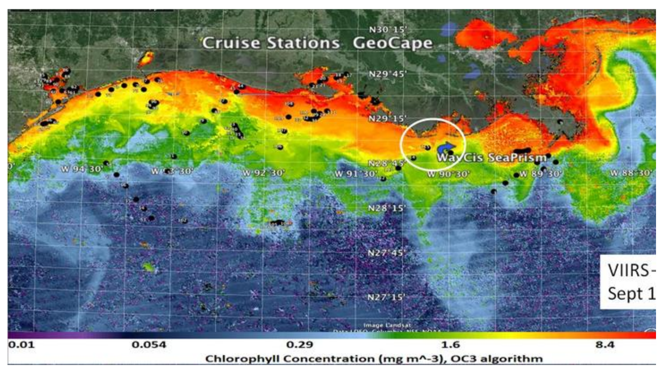
Some recent scheduling applications

Automated robotic cloud in life sciences

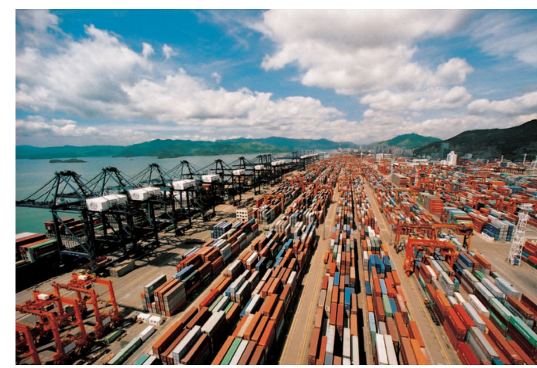
Integrated facility management

Semiconductor wafer fabs

Aircraft assembly

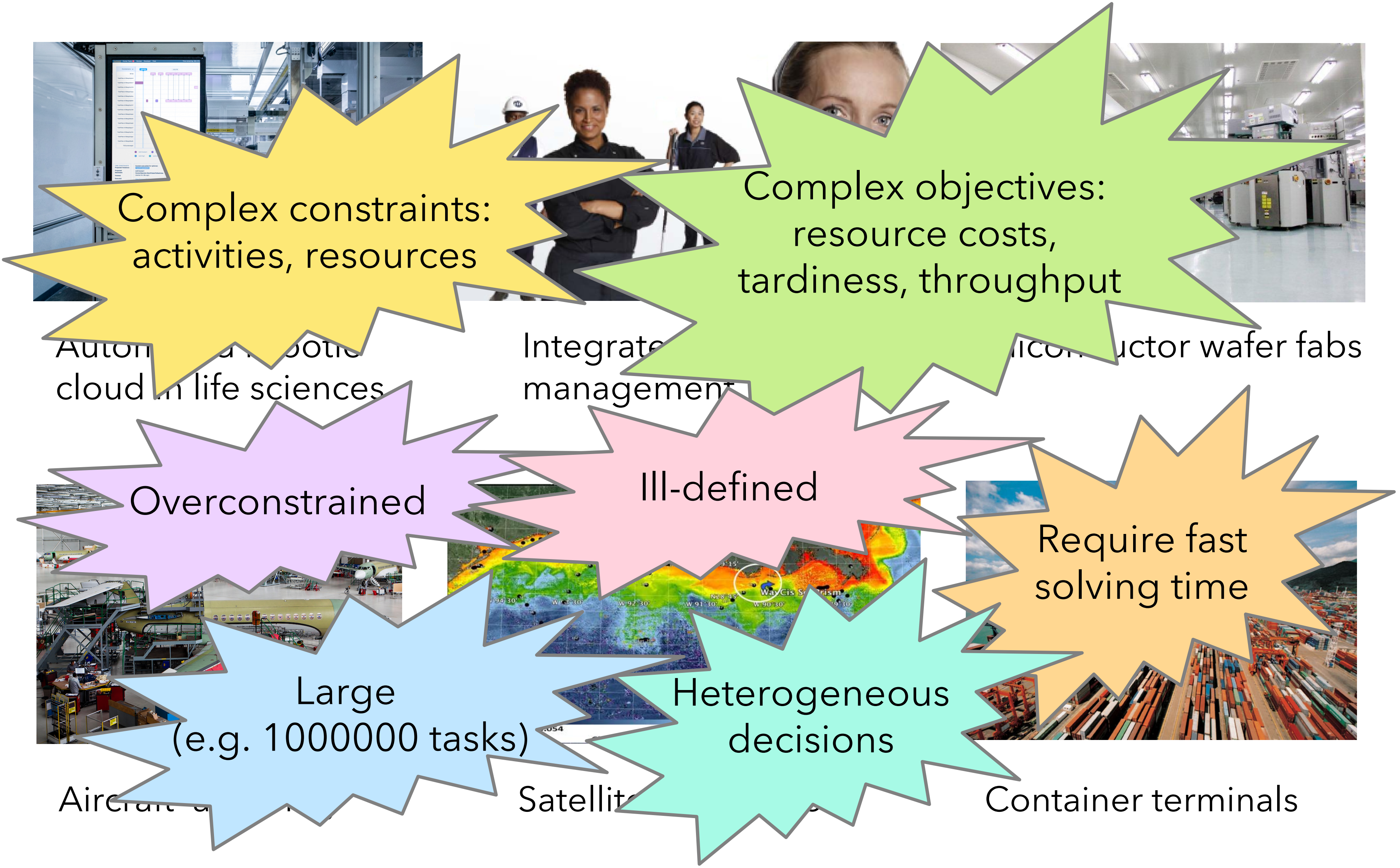


Satellite observations



Container terminals

Some recent scheduling applications



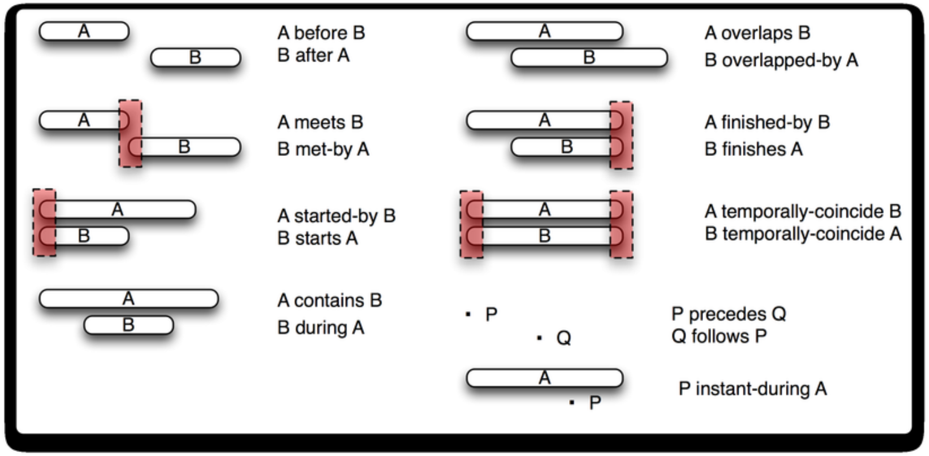
Modeling concepts

Claims:

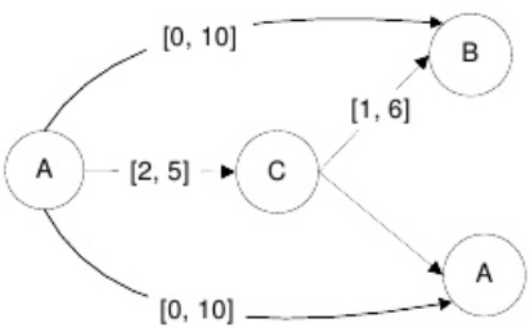
- Both Math Programming (MILP) and classical CP are not using the right abstractions to model scheduling problems
- Numerical decision variables alone (integer, floating point) make it hard to capture the essence and the structure of scheduling problems … (even with a catalogue of more than 400 global constraints in CP)
- It's missing the essential ingredient of scheduling: **time**
- Interestingly, **time** is a very relevant topic in AI
  - Temporal reasoning
  - Reasoning on action and change
  - AI Planning

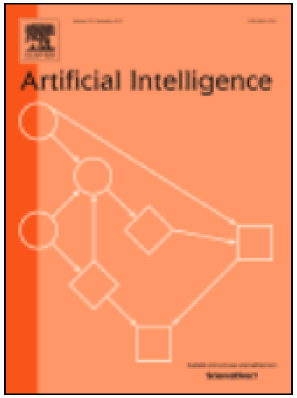
Time in AI: examples

- Allen's interval algebra (1983)



- Temporal constraint networks (1991)

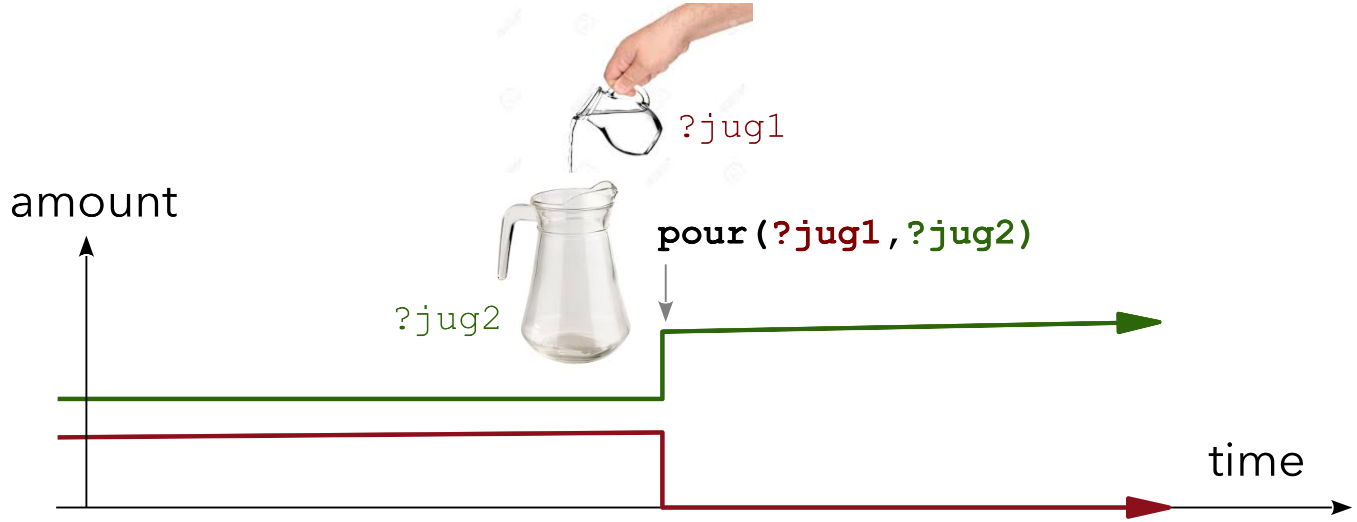




Time in AI: examples

- Temporal planning in PDDL 2.1 (2003)


```
(define (domain jug-pouring)
   (:requirements :typing :fluents)
   (:types jug)
   (:functions
      (amount ?j - jug)
      (capacity ?j - jug))
   (:action pour
      :parameters (?jug1 ?jug2 - jug)
      :precondition (>= (- (capacity ?jug2) (amount ?jug2)) (amount ?jug1))
      :effect (and (assign (amount ?jug1) 0)
                   (increase (amount ?jug2) (amount ?jug1)))
)
```



Time in AI: examples

- MILP and classical CP models only deal with numerical values ($x \in \mathbb{R}$)
- A set of other simple mathematical concepts seem to naturally emerge when dealing with **time**:
  - Intervals : $a = [s,e) = \{ x \in \mathbb{R} \mid s \leq x < e \}$
  - Functions : $f\colon \mathbb{R} \to \mathbb{Z}$
  - Permutations
  - Occurrence / non-occurrence of an event : optional interval

What is CP Optimizer ?

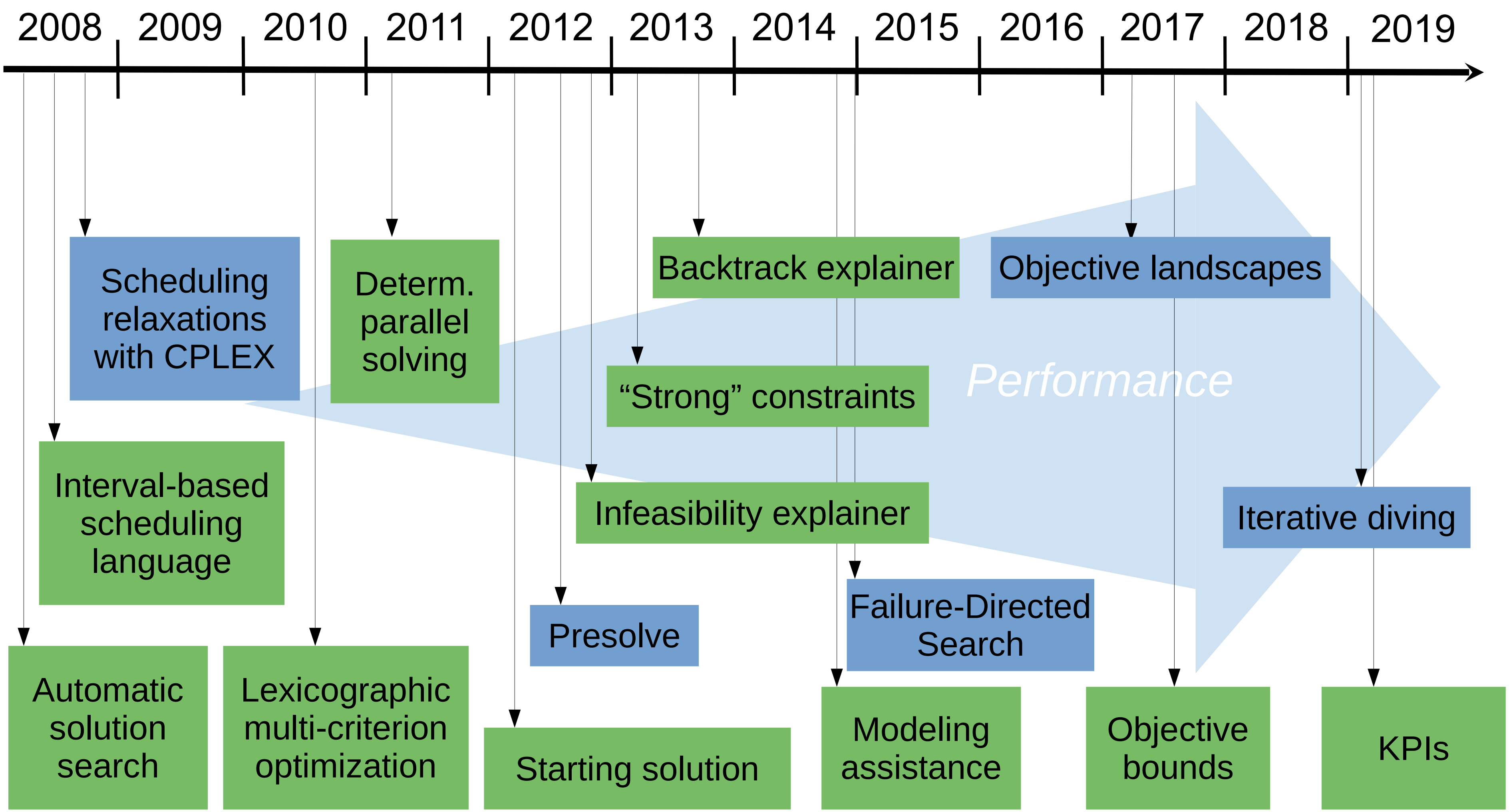
- What if we exploit the flexibility of CP to integrate these mathematical concepts in the model …
- … and use all the good ideas of MILP solvers:
  - Model & run
  - Exact algorithm
  - Input/output file format
  - Language versatility (C++, Python, Java, C#, OPL)
  - Modeling assistance (warnings, …)
  - Conflict refiner
  - Warm-start
  - …
- That's exactly what CP Optimizer is about !

CP Optimizer for scheduling in two sentences

- A **mathematical modeling language** for combinatorial optimization problems that extends MILP (and classical CP) with some algebra on **intervals**, **sequences** and **functions** allowing compact and maintainable formulations for complex scheduling problems

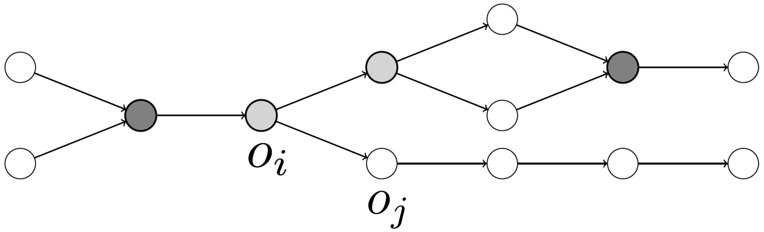
- A continuously improving **automatic search algorithm** that is complete, anytime, efficient (e.g. competitive with problem-specific algorithms on classical problems) and scalable

CP Optimizer timeline



Example: extended flexible job-shop scheduling

- Set of n operations o_i to be scheduled on a set of machines
- Precedence constraints between operations

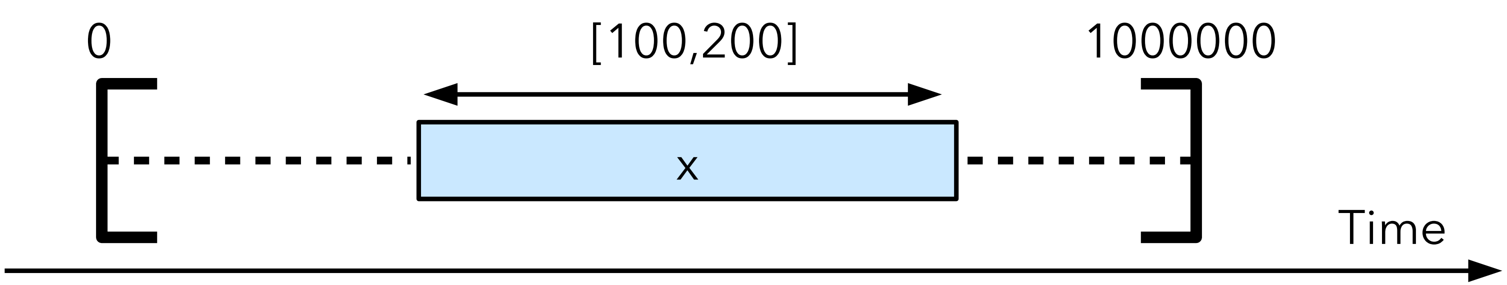


- An operation o_i needs to be allocated on a machine k in a set F_i qualified for executing the operation
- Operation's duration p_{ik} depends on selected machine k
- Machines can only perform one operation at a time
- Objective is to minimize the makespan

CP Optimizer for scheduling – Interval variable

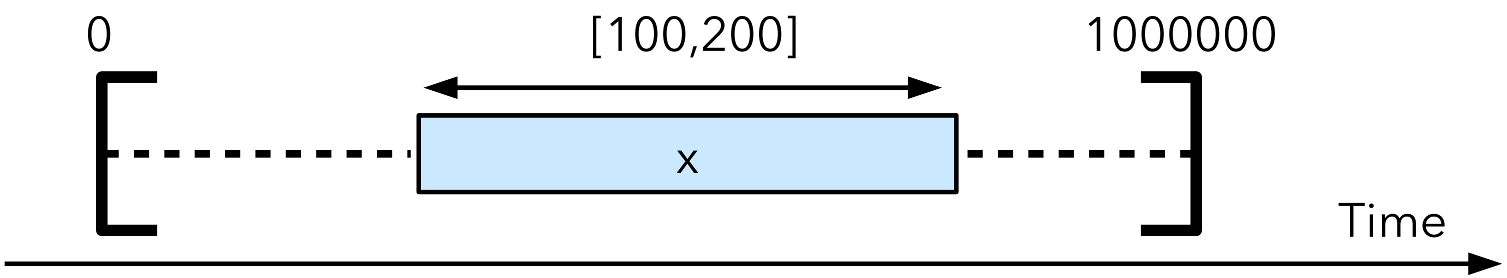
Interval variables

- What for?
  - Modeling an interval of time during which a particular property holds (an activity executes, a resource is idle, a tank must remain empty, ...)
- Example:
  - dvar interval x in 0..1000000 size 100..200;



CP Optimizer for scheduling – Interval variable

```
dvar interval x in 0..1000000 size 100..200;
```



- Properties:
  - The value of an interval variable is an integer interval [start,end)
  - Domain of possible values: [0,100), [1,101), [2,102),... [999900,1000000), [0,101),[1,102),...
  - Domain of interval variables is represented **compactly** inside CP Optimizer (a few bounds: smin, smax, emin, emax, szmin, szmax)

CP Optimizer for scheduling – Interval variable

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..1000000 size in 100..200
- An optional interval variable has an additional possible value in its domain (absence value)

Optional interval variable

- An **optional** interval variable has an additional possible value in its domain (absence value)
- Domain of values for an optional interval variable x:

$$\mathrm{Dom}(x) \subseteq \{\perp\} \cup \{ [s,e) \mid s,e \in \mathbb{Z},\ s \le e \}$$

Absent interval

Interval of integers
(when interval is **present**)

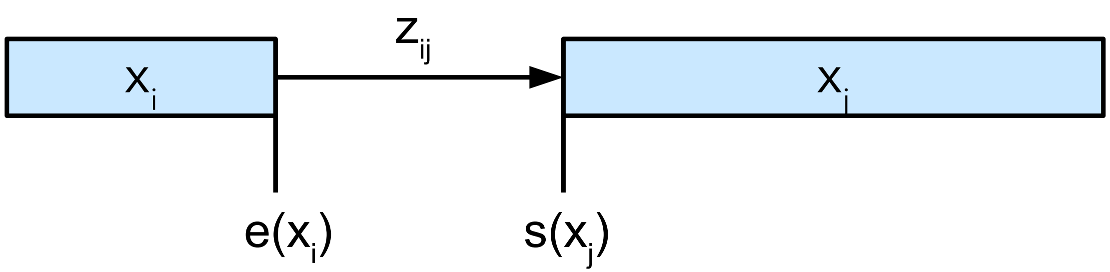
- Constraints and expressions on interval variables specify how they handle the case of absent intervals (in general it is very intuitive)

CP Optimizer for scheduling – Precedence constraints

- Example:

  $\mathrm{endBeforeStart}(x_i, x_j, z_{ij})$ means:

  $(x_i \neq \perp) \wedge (x_j \neq \perp) \Rightarrow e(x_i) + z_{ij} \leq s(x_j)$



CP Optimizer for scheduling – Logical constraints

- Unary presence constraint

  presenceOf(x) means: $(x \neq \perp)$

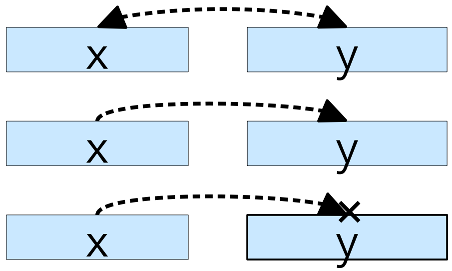
- Logical binary constraints between presence status:

  Examples:

  presenceOf(x) == presenceOf(y)

  presenceOf(x) => presenceOf(y)

  presenceOf(x) => !presenceOf(y)



- Of course, other combinations are also possible

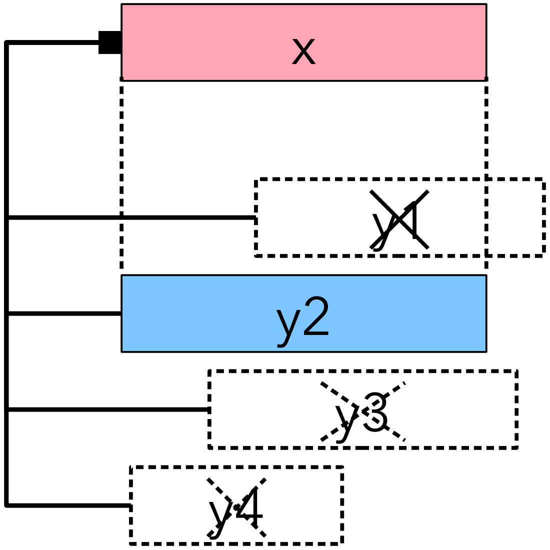
  presenceOf(x) && presenceOf(y) =>
  presenceOf(u) || presenceOf(v)

CP Optimizer for scheduling – Alternative constraint

- Alternative constraint:

  alternative(x, [y1,…,yn])

- If *x* is present, then exactly one of the {*y1*,…,*yn*} is present and synchronized with *x* (same start and end value)
- If *x* is absent, then all *yi* are absent too

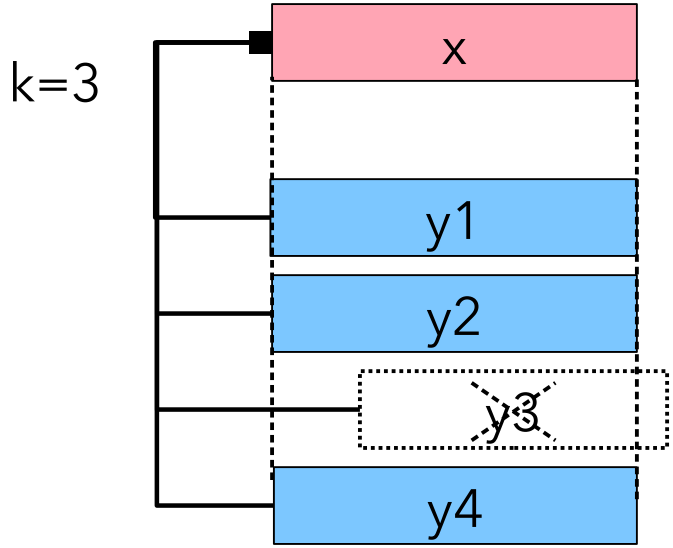


CP Optimizer for scheduling – Alternative constraint

- Generalized alternative constraint

  alternative(x, [y1,…,yn], k) k: integer expression

- If *x* is present, then exactly *k* of the {*y1*,…,*yn*} are present and synchronized with *x* (same start and end value)
- If *x* is absent, then all *yi* are absent too



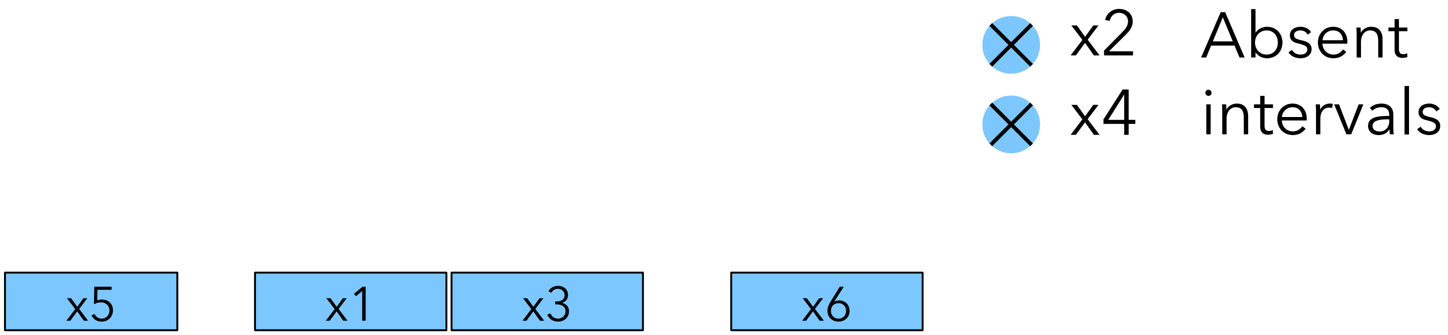
CP Optimizer for scheduling – No-overlap constraint

- No-overlap constraint

  noOverlap([x1,...,xn])

- The set of present intervals in {x*1*,...,x*n*} do not overlap

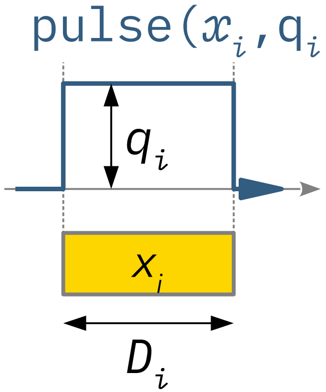
  noOverlap([x1,...,x6])



CP Optimizer for scheduling – Cumul functions

- A cumul function is the sum of elementary functions like pulse, stepAtStart, stepAtEnd

  $f = \Sigma_i \text{ pulse}(x_i,q_i)$



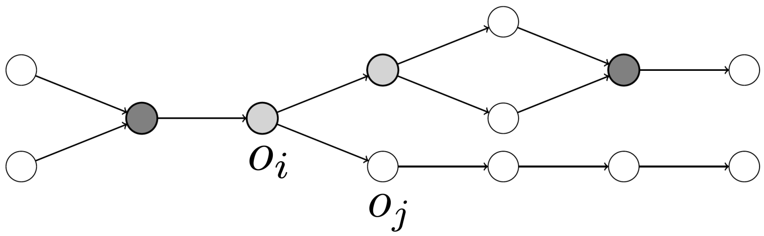
- The value of a cumul function is a stepwise function
- Constraints can be posted on cumul functions:

  $f \leq C$

  alwaysIn(f,x,levelMin,levelMax)

Example: extended flexible job-shop scheduling

- CP Optimizer formulation:



$$
\begin{aligned}
\text{Minimize } \max_{i \in V} \text{endOf}(o_i) & \\
\text{subject to} & \\
\text{endBeforeStart}(o_i, o_j) & \quad \forall (i,j) \in A, \\
\text{alternative}(o_i, [a_{ik}]_{k \in F_i}) & \quad \forall i \in V, \\
\text{noOverlap}([a_{ik}]_{i \in V : k \in F_i}) & \quad \forall k \in [1, m], \\
\text{interval } o_i & \quad \forall i \in V, \\
\text{interval } a_{ik}, \text{ optional, size} = p_{ik} & \quad \forall i \in V, k \in F_i
\end{aligned}
$$

Example: extended flexible job-shop scheduling

```
# Create model object
model = CpoModel()

# Decision variables: o[i] is operation i
o = [interval_var() for i in V ]

# Decision variables: a[i][k] is operation i on machine k
a = [{k:interval_var(optional=True,size=p) for k,p in F[i]} for i in V]

# Objective: minimize makespan
model.add(minimize(max(end_of(o[i]) for i in V)))

# Constraints: precedence constraints
model.add([end_before_start(o[i],o[j]) for i,j in A])

# Constraints: machine allocation
model.add([alternative(o[i], a[i].values()) for i in V])

# Constraints: machines are unary resources
model.add([no_overlap([a[i][k] for i in V if k in a[i]]) for k in M])

# Solve the model
sol = model.solve(trace_log=True, LogPeriod=1000000)
```

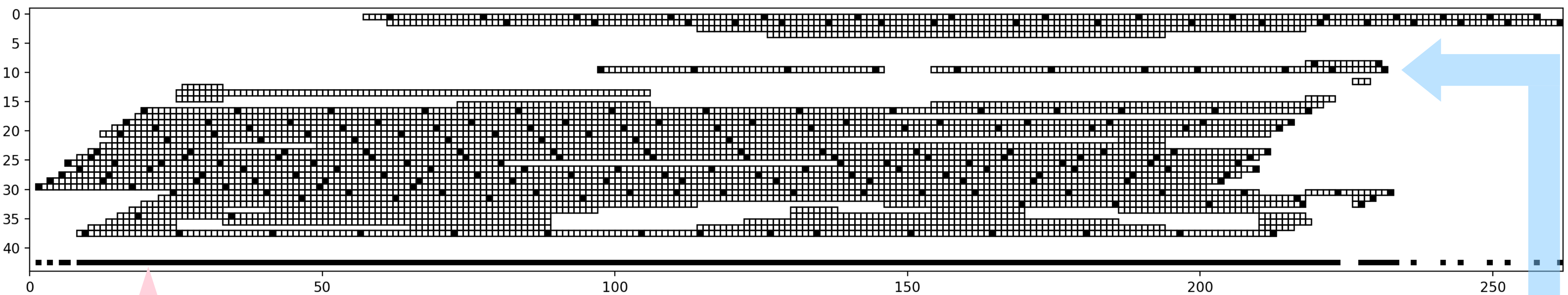
CP Optimizer for scheduling

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

Features	Concept	Features
Earliness/tardiness costs	Constant functions	Resource calendars Resource efficiency
Temporal constraints Optional activities Over-constrained problems Alternative resources/modes Work-breakdown structures	Interval variables	
	Sequence variables	Unary resources Setup times/costs Travel times/costs
	Cumul functions	Cumulative resources Inventories, Reservoirs
	State functions	Parallel batches Activity incompatibilities
Aggregation of individual costs (max, weighted sum, Net Present Value)	General arithmetical expressions	

Example: a satellite observation scheduling problem

- A problem instance and a feasible solution (ICAPS-2007)



A scene with its possible observability time-slots

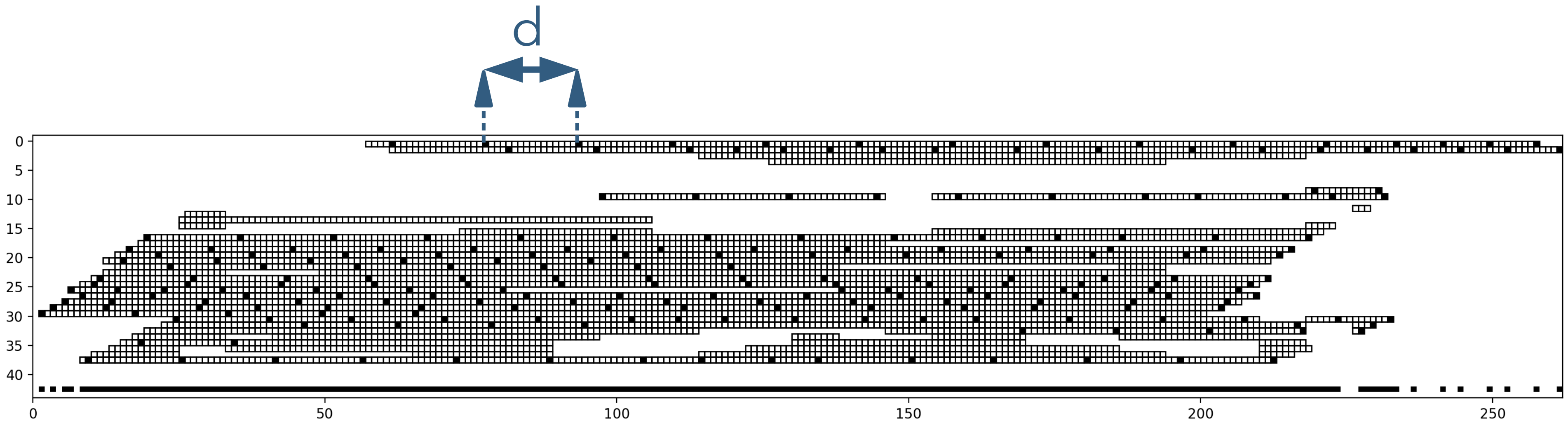
→ Observability time-slots

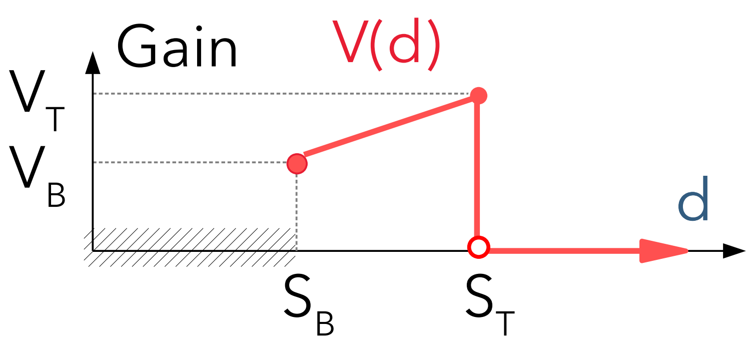
→ Scheduled observations

All scheduled observations

Example: a satellite observation scheduling problem

- Schedule quality depends on the separation time *d* between consecutive observations of each scene





- Objective: **maximize** total gain due to separation times
- Number of scheduled observations is unknown

Example: a satellite observation scheduling problem

```
using CP;
int   SB = ...; int   ST = ...;
float VB = ...; float VT = ...;
float A = (VT-VB) / (ST-SB);
int n = ...;
{int} T[1..n] = ...;
int TL = min(i in 1..n, t in T[i]) t;
int TU = max(i in 1..n, t in T[i]) t;
int m = (TU-TL) div SB;

pwlFunction V = piecewise{ A->ST; -VT->ST+1; 0 } (SB,VB);
stepFunction NoObs[i in 1..n] =
  stepwise(t in TL-1..TU) { (t in T[i]) -> t+1; 0 };

dvar interval a [1..n, 1..m+1] optional size 1;
dvar interval s [1..n, 1..m]   optional;
dvar interval sv[1..n, 1..m]   optional size SB..ST;
dvar interval s0[1..n, 1..m]   optional size ST+1..TU;

maximize sum(i in 1..n, j in 1..m) lengthEval(s[i,j], V);
subject to {
  forall(i in 1..n, j in 1..m+1) {
    if (j < m+1) {
      presenceOf(a[i,j+1]) == presenceOf(s[i,j]);
      startAtStart(a[i,j], s[i,j]);
      endAtStart(s[i,j], a[i,j+1]);
      alternative(s[i,j], append(sv[i,j], s0[i,j]));
      if (j == 1) {
        presenceOf(a[i,j+1]) == presenceOf(a[i,j]);
        !presenceOf(s0[i,j]);
      } else {
        presenceOf(a[i,j+1])  => presenceOf(a[i,j]);
        presenceOf(s0[i,j-1]) => presenceOf(sv[i,j]);
      }
    }
    forbidExtent(a[i,j], NoObs[i]);
  }
  noOverlap(a);
}
```

Data reading and constants

Decision variables

Objective function

Constraints

CP Optimizer automatic search

- In OPL IDE: Press the **`solve`** button !
- In the other APIs: Call a function **`solve()`** !

CP Optimizer automatic search - Properties

- Search is **complete**
- Search is **anytime**
- Search is **parallel** (unless stated otherwise)
- Search is **randomized**
  - Internally, some ties are broken using random numbers
  - The seed of the random number generator is a parameter of the search
- Search is **deterministic**
  - Solving twice the same problem on the same machine (even when using multiple parallel workers) with the same seed for the internal random number generator will produce the same result
  - Determinism of the search is **essential** in an industrial context and for debugging

CP Optimizer automatic search - Under the hood

Artificial Intelligence **Operations Research**

Heuristics

Model presolve

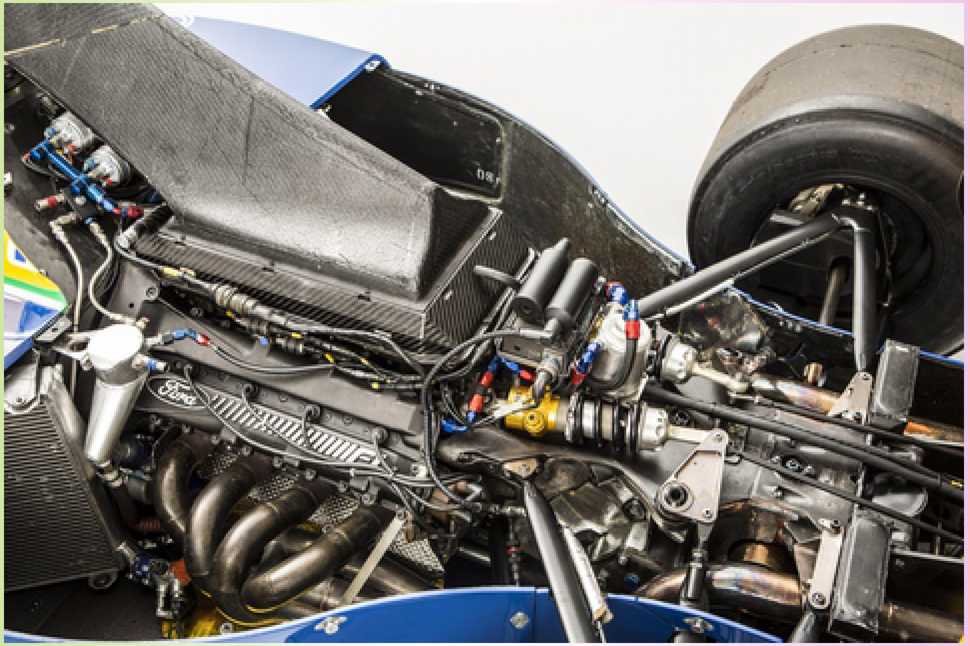
Constraint propagation

Learning

Temporal constraint networks

2-SAT networks

No-goods



Linear relaxations

Problem specific scheduling algorithms

Restarts

Tree search

LNS

Randomization

CP Optimizer automatic search

Constraint Propagation: Logical network

- Aggregates all binary constraints on interval presence as an implication graph between literals or their opposite

  [!]presenceOf(u) ⇒ [!]presenceOf(v)

- Equivalent to a 2-SAT model
- Computes graph condensation and transitive closure:
  - Detects infeasibility
  - Allows querying in O(1) whether [!]presenceOf(x) ⇒ [!]presenceOf(y) for any (x,y)

CP Optimizer automatic search

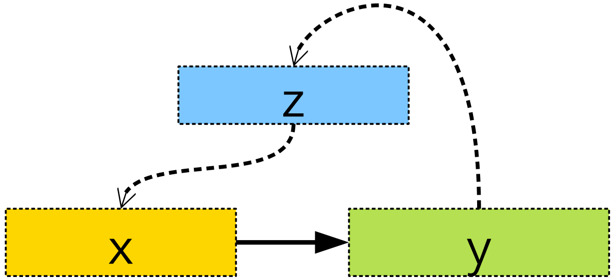
Constraint Propagation: Precedence network

- Aggregates all precedence constraints (like endBeforeStart(u,v,d_{uv})) in a *Simple Temporal Network (STN)* extended with Boolean presence status of nodes
  - Nodes: start or end of (optional) interval variables
  - Arcs: minimal delay between two nodes
- Temporal domain of a node *t* is maintained as a range $[t_{min},t_{max}]$ representing the possible values **if the interval is present**
- Propagation exploits the **Logical network**

Constraint Propagation: Precedence network

- Propagation exploits the **Logical network**
- Example:

  endBeforeStart(x,y)
  presenceOf(y)=>presenceOf(z)
  presenceOf(z)=>presenceOf(x)



- Logical network deduces:

  presenceOf(y)=>presenceOf(x)

- The precedence constraint can propagate the bounds of *x* on *y* : `smin(y)` $\leftarrow$ `max( smin(y), emin(x) )`
- This is very powerful: propagation occurs even when the presence status is still unfixed

CP Optimizer automatic search

Constraint Propagation: Precedence network

- Propagation exploits the **Logical network**
- This is very powerful: propagation occurs even when the presence status is still unfixed
- Classical STN propagation algorithms are extended to perform this type of directional propagation
- Algorithms used in CP Optimizer:
  - At root node: extension of an improved version of Bellman-Ford algorithm: B. Cherkassky, A. Goldberg, T. Radzic. *Shortest Paths Algorithms: Theory and Experimental Evaluation.* Mathematical Programming 73, 129–174 (1996)
  - During the search: extension of the algorithm described in: A. Cesta, A. Oddi. *Gaining Efficiency and Flexibility in the Simple Temporal Problem.* In: Proc. TIME-96 (1996)

CP Optimizer automatic search

Constraint Propagation: Timelines (noOverlap, cumulFunction, ...)

- Default propagation algorithm is the **timetable** that incrementally maintains the domain of the function as a set of segments with bounds on the function values
- Sequence variables (noOverlap) are internally represented as a **precedence graph** on interval variables

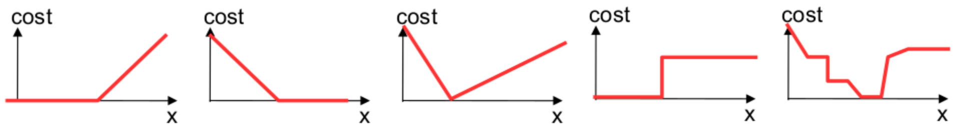
CP Optimizer automatic search

Constraint Propagation: Timelines (noOverlap, cumulFunction, ...)

- Stronger propagation algorithms are available and are automatically turned on in the search **depending on the context**:
  - Multiple O(n log(n)) algorithms for disjunctions (noOverlap): P. Vilím: *Global constraints in scheduling*. Ph.D. thesis. 2007.
  - O(n^2) time-table edge-finding for cumul functions: P. Vilím: *Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources*. Proc. CPAIOR-2011.

Linear Relaxation

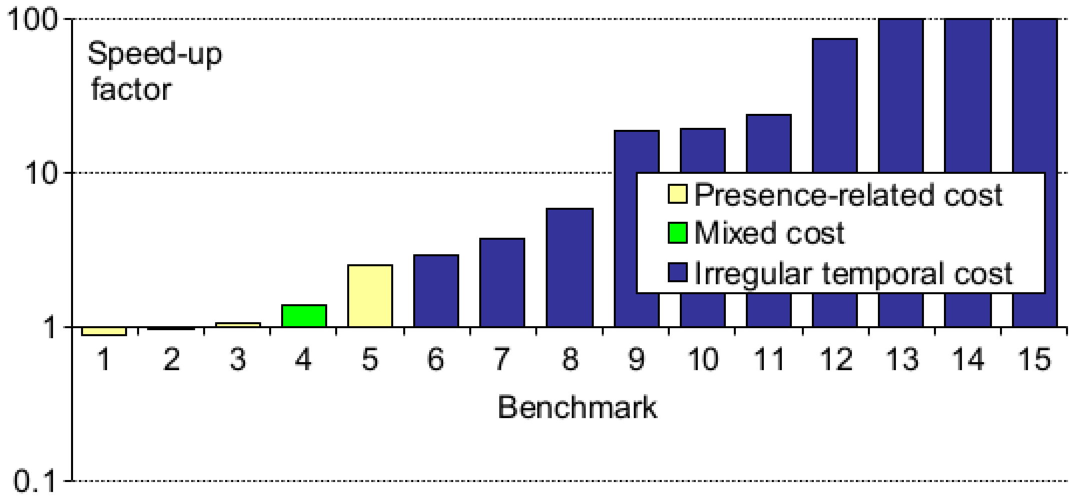
- CPLEX LP is used to provide a relaxation to the scheduling (sub-)problem
- What is relaxed?
  - Irregular cost functions



  - Precedences, logical constraints between intervals
  - Alternatives, spans
  - Linear constraints in the formulation
  - Etc.
- Result is used:
  - For computing lower-bounds
  - As a heuristic to guide the search

Linear Relaxation

- Results



CP Optimizer automatic search

Presolve

- Model analysis before search begins. The hope is to improve the formulation by replacing constructs with others that will be more efficient
- Transforms the initial model into a new one. The transformed model is fed to the workers
- CP Optimizer does different types of operations inside presolve
  - Basic simplifications. e.g. constant propagation, linear simplification
  - Aggregation / combination. e.g. common sub-expression elimination
  - Higher level transformations

CP Optimizer automatic search

Presolve (examples)

- startOf(y) ≤ endOf(x) → endBeforeStart(x, y)
- endOf(x) – startOf(x) → lengthOf(x)
- startBeforeStart(x, y) ∧ noOverlap([…,x,…,y,…])

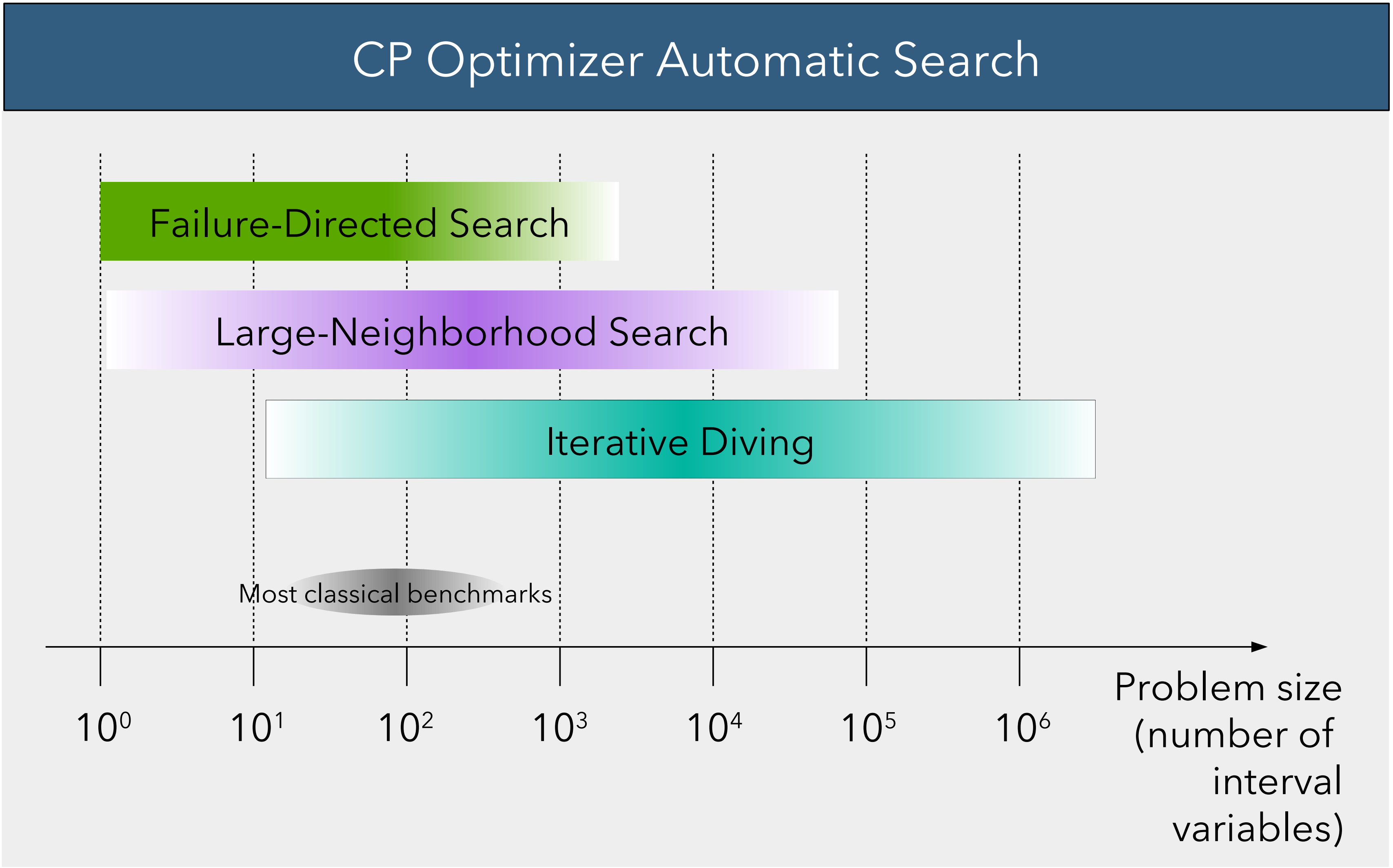
  → endBeforeStart(x, y)
- presenceOf(x) - presenceOf(y) <= 0

  → presenceOf(x) ⇒ presenceOf(y)
- x!=y, y!=z, x!=z

  → allDifferent([x,y,z])

CP Optimizer automatic search

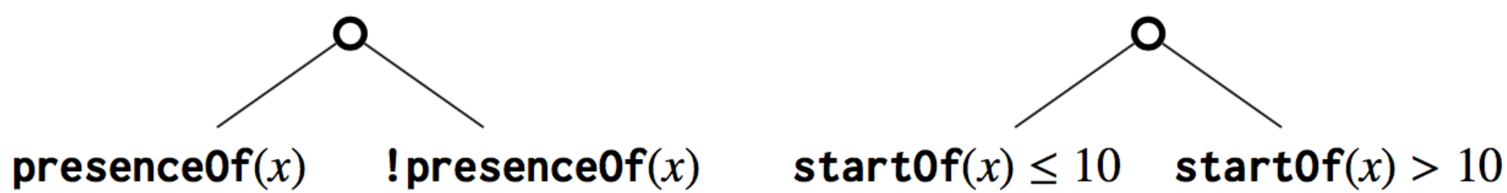
- Main principle: cooperation between several approaches



CP Optimizer automatic search

Failure-Directed Search (FDS) (complete search)

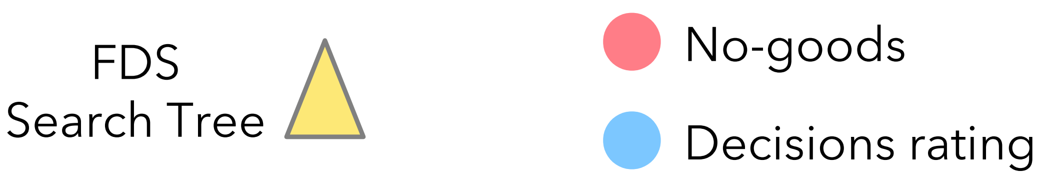
- FDS is automatically activated when:
  - The search space seems to be small enough, and
  - LNS has difficulties improving the current solution
- Assumption is that in these conditions:
  - There probably isn't any (better) solution
  - If there is one, it is very hard to find
  - It is necessary to explore the whole search space
- FDS uses periodic restarts and focuses on finding dead-ends (failures) in the search tree as quickly as possible
- FDS branches on ranges

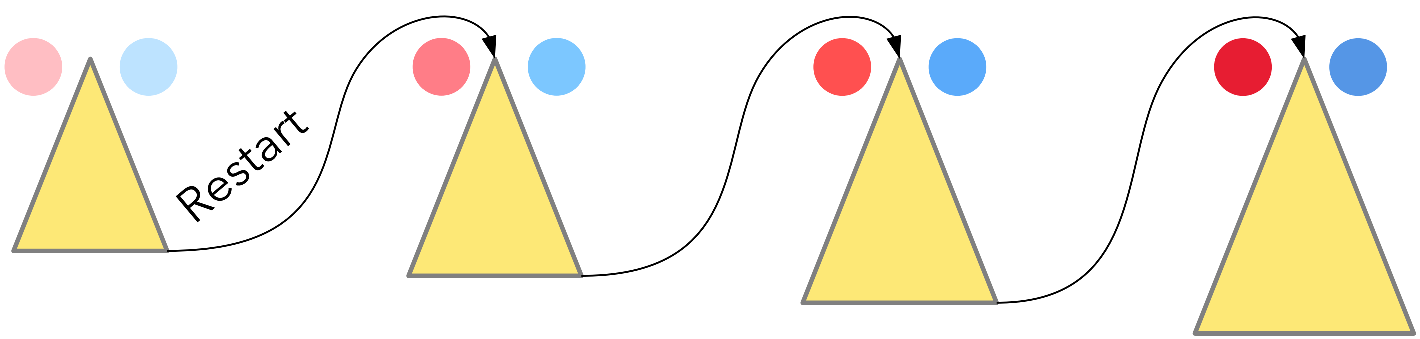


CP Optimizer automatic search

Failure-Directed Search (FDS) (complete search)

- Decisions are rated and the ones that often lead to strong domain reduction in the search are preferred: they are used earlier in the search during the next restarts
- FDS also records no-goods for avoiding exploring some identical part of the search space

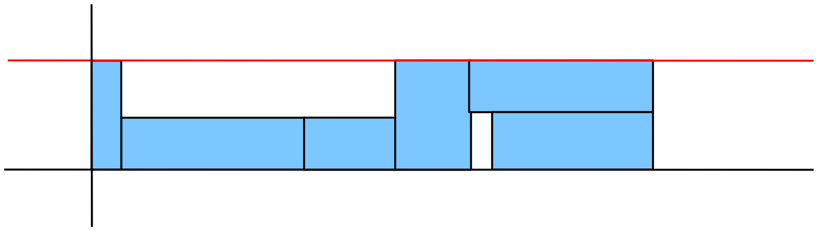




CP Optimizer automatic search

Large-Neighborhood Search (LNS) (heuristic search)

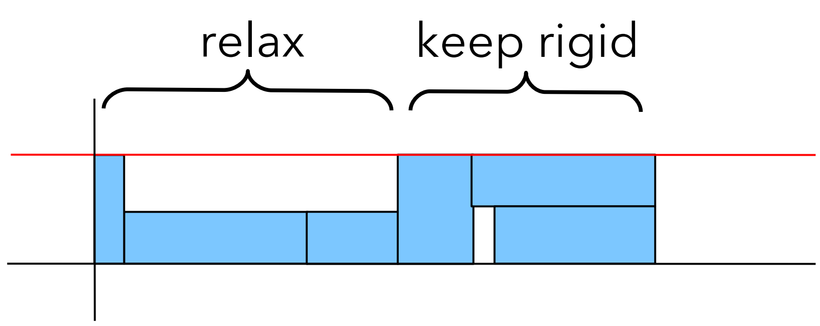
- Iterative improvement method:
  1. Start with an existing solution (produced using some heuristics + classical CP search tree)



CP Optimizer automatic search

Large-Neighborhood Search (LNS) (heuristic search)

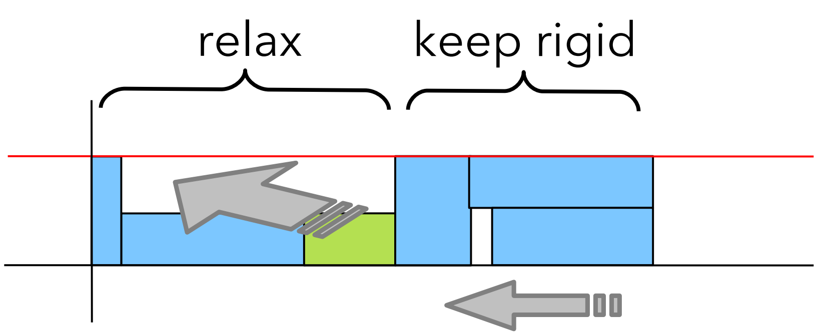
- Iterative improvement method:
  1. Start with an existing solution (produced using some heuristics + classical CP search tree)
  2. Take part of the solution (fragment) and relax it. Fix the structure of the rest (but no start/end values: notion of **Partial Order Schedule**)



CP Optimizer automatic search

Large-Neighborhood Search (LNS) (heuristic search)

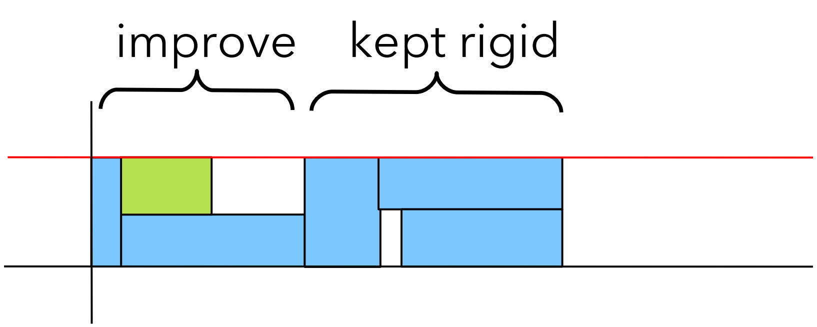
- Iterative improvement method:
  1. Start with an existing solution (produced using some heuristics + classical CP search tree)
  2. Take part of the solution (fragment) and relax it. Fix the structure of the rest (but no start/end values: notion of **Partial Order Schedule**)
  3. Find (improved) solution using a limited search tree



CP Optimizer automatic search

Large-Neighborhood Search (LNS) (heuristic search)

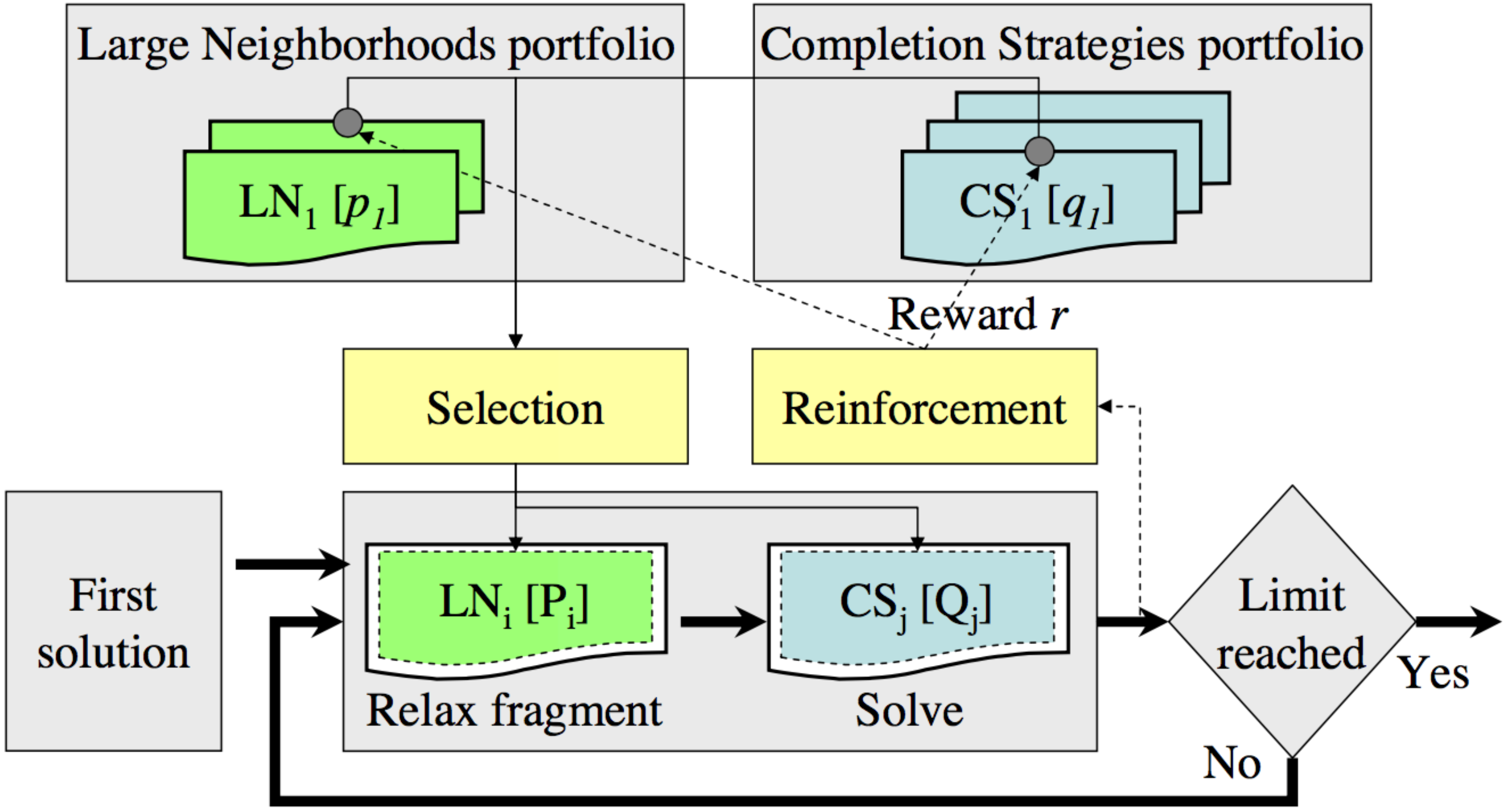
- Iterative improvement method:
  1. Start with an existing solution (produced using some heuristics + classical CP search tree)
  2. Take part of the solution (fragment) and relax it. Fix the structure of the rest (but no start/end values: notion of **Partial Order Schedule**)
  3. Find (improved) solution using a limited search tree



CP Optimizer automatic search

Large-Neighborhood Search (LNS) (heuristic search)

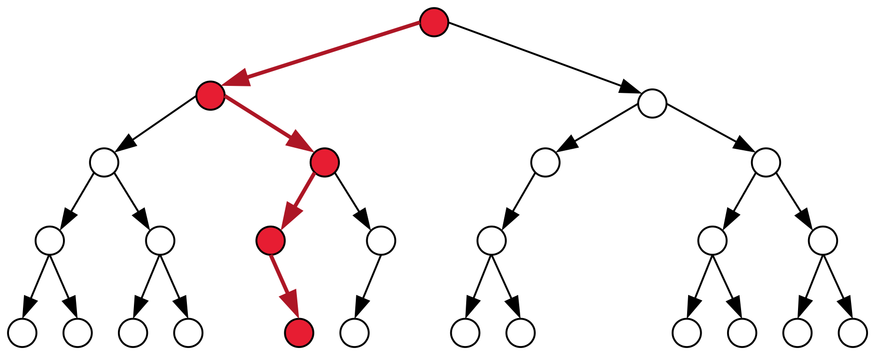
- Uses portfolios and online reinforcement learning



CP Optimizer automatic search

Iterative Diving (heuristic search)

- Idea of iterative diving: perform aggressive dives (no backtrack) in the search tree explored by CP

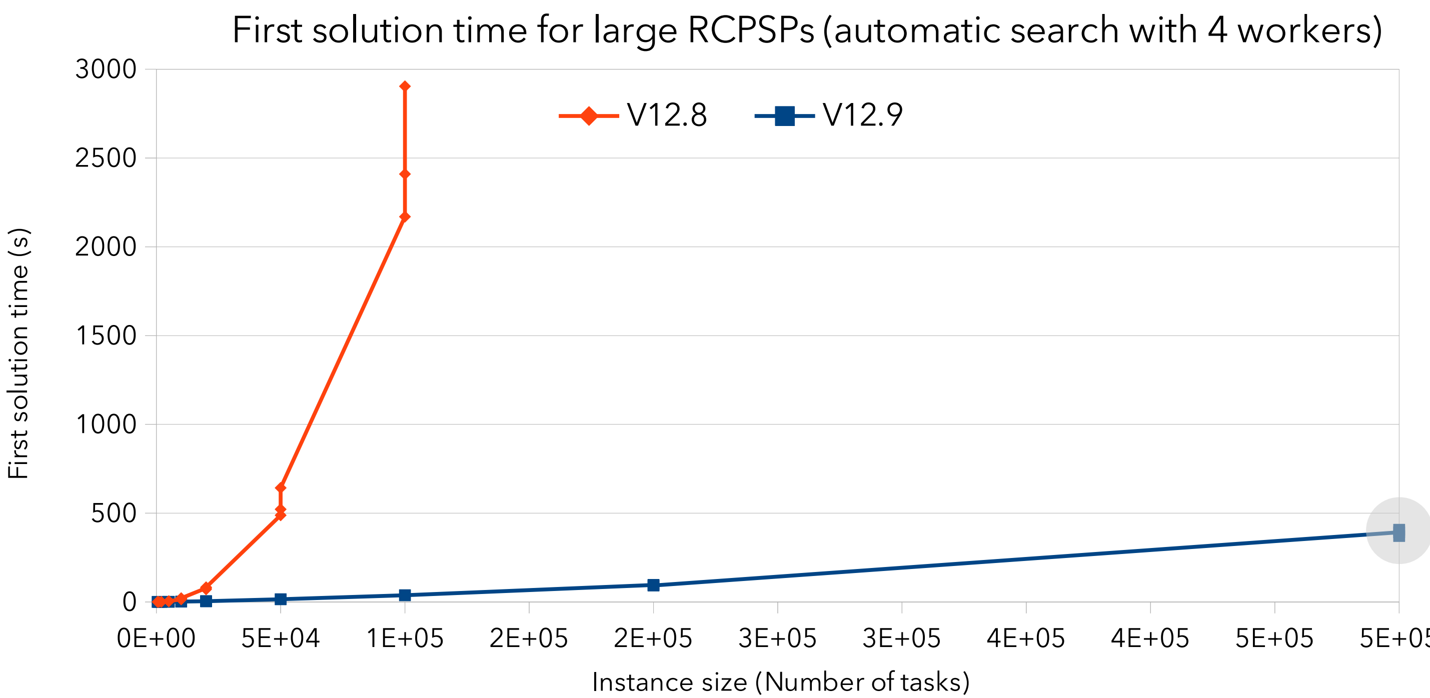


- For instance on RCPSP it boils down to some very classical ideas (list scheduling, decoding schemes, ...)
- The challenge is to generalize these problem-specific ideas to the general modeling concepts of CP Optimizer

CP Optimizer automatic search

Iterative Diving (heuristic search)

- New benchmark with RCPSP from 500 to 500.000 tasks
  - Largest problem: 500.000 tasks, 79 resources, 4.740.783 precedences, 4.433.550 resource requirements
- Time to first feasible solution (V12.8 v.s. V12.9)



CP Optimizer automatic search - Performance

- Results published in CPAIOR-2015 (using V12.6)
  - Job-shop
    - 15 instances closed out of 48 open ones
  - Job-shop with operators
    - 208 instances closed out of 222 open ones
  - Flexible job-shop
    - 74 instances closed out of 107 open ones
  - RCPSP
    - 52 new lower bounds + 39 instances closed in 2019
  - RCPSP with maximum delays
    - 51 new lower bounds out of 58 small-medium instances
      + 372 bounds improved on large instances in 2019
  - Multi-mode RCPSP
    - 535 instances closed out of 552
  - Multi-mode RCPSP with maximum delays
    - All 85 open instances of the benchmark closed

Performance evaluation

- As of today, our performance evaluation benchmark contains 159 different scheduling models tested on a total of 3436 instances

#	Family	Tests	Type
	Total / Average	3436	Mxd
1	A380Maintenance	7	Min
2	ABCProblem	9	Min
3	Acid	1	Max
4	AircraftAssemblyDS	2	Min
5	AircraftAssemblyShifts	1	Min
6	AircraftLines	7	Opt
7	AircraftMaintenance	1	Min
8	AircraftParts	1	Min
9	AirLand	25	Min
10	AirportGates	1	Min
11	AirTrafficFlowManagement	1	Min
12	AssemblyWithInventories	3	Min
13	AuditScheduling	40	Min
14	BlockingJobShop	5	Min
15	BreakScheduling	1	Min
16	Cargo	15	Min
17	Carpet-cutting	10	Min
18	Cities	1	Max
19	Coils_CoilBatching	1	Min
20	Coils_Global_CoilBatching	1	Min
21	CommonDueDate	20	Min
22	CoupledTasks	4	Min
23	CP2015Competition	14	Min
24	CraneSchedulingAverage	14	Min
25	CraneSchedulingDifficult	19	Min
26	CropScheduling	1	Min
27	CumulativeJobShop	38	Min
28	CVRPTW	29	Min
29	Cyclic-rcpsp	5	Min
30	DetailedProjectScheduling	4	Min
31	DevProject	1	Min
32	Diffusion	10	Min
33	DisjunctiveGraph	2	Min
34	Doors	1	Min
35	DynamicResourceFeasibility	18	Min
36	EarthObservationSatellite	1	Max
37	Filters	15	Min
38	FishBoats	2	Min
39	Fjsp	5	Min
40	FLETC	1	Min
41	FlexibleJobShop	56	Min
42	FlowShop	12	Min
43	FlowShopBuffers	30	Min
44	FlowShopEarliTardi	12	Min
45	FlowShopMinMax	6	Min
46	GEOCAPEObservationScheduling	15	Max
47	Gfd-schedule	10	Min
48	Ghoulomb	5	Min
49	HoistScheduling	10	Min
50	Hooker	15	Min
51	HookerCost	15	Min
52	HugeFlexibleJobShop	4	Min
53	HugeHookerCost	7	Min
54	HugeJobShop	4	Min
55	HugeOpenShop	4	Min
56	HugeRCPSP	7	Min
57	HugeSingleCumulative	4	Min
58	HugeSingleNoOverlap	3	Min
59	Inspectors	1	Min
60	InstructionScheduling	1	Min
61	JobShop	34	Min
62	JobShopEarliTardi	48	Min
63	JobShopEnergy	10	Min
64	JobShopEnergyLimit	6	Min
65	JobShopOperators	5	Min
66	JobShopOperatorsFlowTime	5	Min
67	JobShopTotalFlowTime	15	Min
68	LargeRCPSP	6	Sat
69	LargeScheduling	8	Min
70	Largescheduling	10	Min
71	LargeShopScheduling	6	Sat
72	LargeTimeNet	3	Sat
73	MaintenanceScheduling	1	Min
74	ManpowerScheduling	1	Max
75	Manufacturing	1	Max
76	MinPeak	1	Min
77	MISTA2013Challenge	10	Min
78	MixedCriticalityMatchUp	16	Min
79	MMASP	34	Min
80	MMRL	30	Min
81	Mrcpsp	5	Min
82	Mspsp	6	Min
83	MultiModeRCPSP	402	Min
84	MultiModeRCPSPMax	54	Min
85	MultiprocessorMakespan	2	Min
86	MultiProcessorTotalTardiness	20	Min
87	MultiprocessorWithCommunicationDelays	60	Min
88	MultiSkillsRCPSP	20	Min
89	MultiStageHybridFlowShop	20	Min
90	NewProductsTesting	8	Max
91	NuclearLabScheduling	10	Min
92	OpenShop	28	Min
93	Openshop	5	Min
94	Oversubscribed	150	Min
95	Pallets	1	Min
96	ParallelMachines	1	Max
97	ParallelMachinesDates	1	Min
98	PathSelection	1	Max
99	PatientScheduling	1	Min
100	PermutationFlowShop	20	Min
101	PermutationFlowshopMaintenanceEarliTardi	10	Min
102	PermutationFlowshopMaintenanceMakespan	10	Min
103	Photolithography	1	Min
104	PickupDelivery	1	Max
105	PPOSingleMachine	116	Min
106	ProductionBatches	1	Min
107	ProductionPlanning	2	Min
108	ProductionWithAlternatives	1	Min
109	ProductionWithCalendars	1	Min
110	Profiles	18	Min
111	QMRCPSP	400	Min
112	QuarriesScheduling	1	Min
113	QuayCraneScheduling	12	Min
114	Racp	5	Min
115	RCPSP	440	Min
116	Rcpsp-wet	10	Min
117	RCPSPDC	50	Max
118	RCPSPEarliTardi	65	Min
119	RCPSPImbalance	1	Min
120	RCPSPMax	80	Min
121	RCPSPMaxCal	6	Min
122	RCPSPMaxInventories	12	Min
123	Rcpsp	5	Min
124	Rectangle-packing	5	Sat
125	ResourceAvailabilityCost	30	Min
126	RideSharing	18	Min
127	SchedSCS	4	Min
128	SchedulingGeneralTimeLags	10	Min
129	SelfPlanner	70	Max
130	SemiconductorTesting	18	Min
131	SequenceDates	1	Min
132	SequenceSetupTimes	1	Min
133	SetupMinimizer	1	Min
134	SingleMachineEarliTardi	40	Min
135	SingleMachineTardiTasks	100	Min
136	SkilledOperators	14	Min
137	Smelt	5	Min
138	SMSDST	16	Min
139	SPLC	10	Min
140	StressedJobShop	15	Min
141	TankTruckScheduling	1	Min
142	Test-scheduling	5	Min
143	TimeTabling	1	Min
144	TrainingBatch	2	Min
145	TrainingProject	3	Min
146	TrainingSatellite	1	Max
147	TrainingSelfPlanner	1	Max
148	TrainingTransportation	2	Opt
149	Trolley	15	Min
150	TruckingWood	1	Min
151	TruckScheduling	1	Min
152	TruckSchedulingFlexibleShifts	1	Min
153	TSP	101	Min
154	UnaryAlternativeTransitionTime	39	Min
155	UPMST	10	Min
156	Voestalpine	1	Min
157	VRPBalancedVehicles	7	Min
158	Vrplc	5	Min
159	YogurtScheduling	10	Min

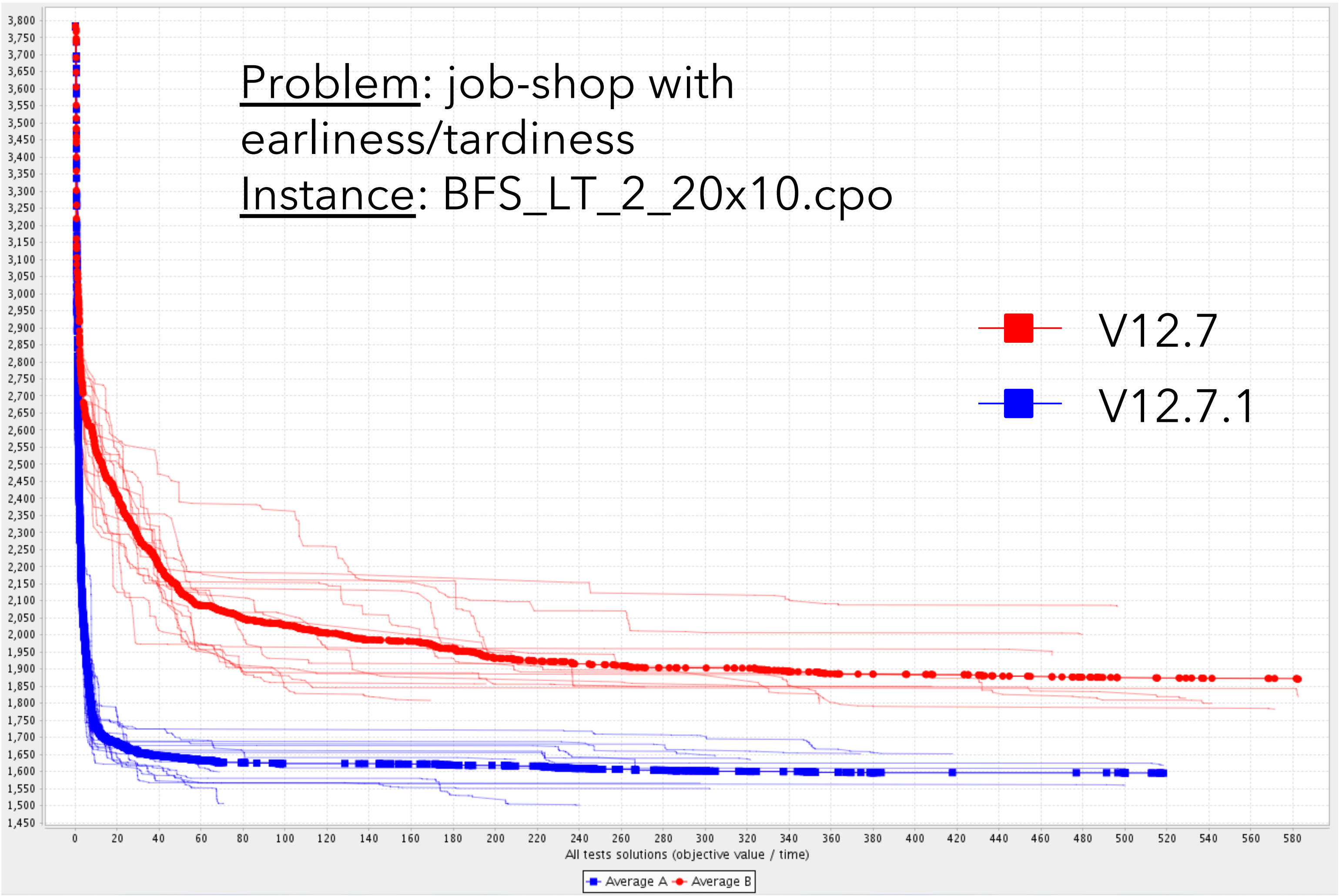
Performance evaluation

- The benchmark collects problems from different sources:
  - Classical problems (job-shop, RCPSP, …)
  - New problems proposed in academic papers
  - Industrial scheduling problems from:
    - Customers
    - Business partners
    - End users
  - Problems discussed on our Optimization forum
  - …
- Problems are quite diverse
  - Size range: 30 to 1.000.000 interval variables
  - Resource types: disjunctive, cumulative
  - Objective functions: makespan, weighted earliness/tardiness, resource allocation costs, activity non-execution penalties, resource transition costs, ...

Performance evaluation

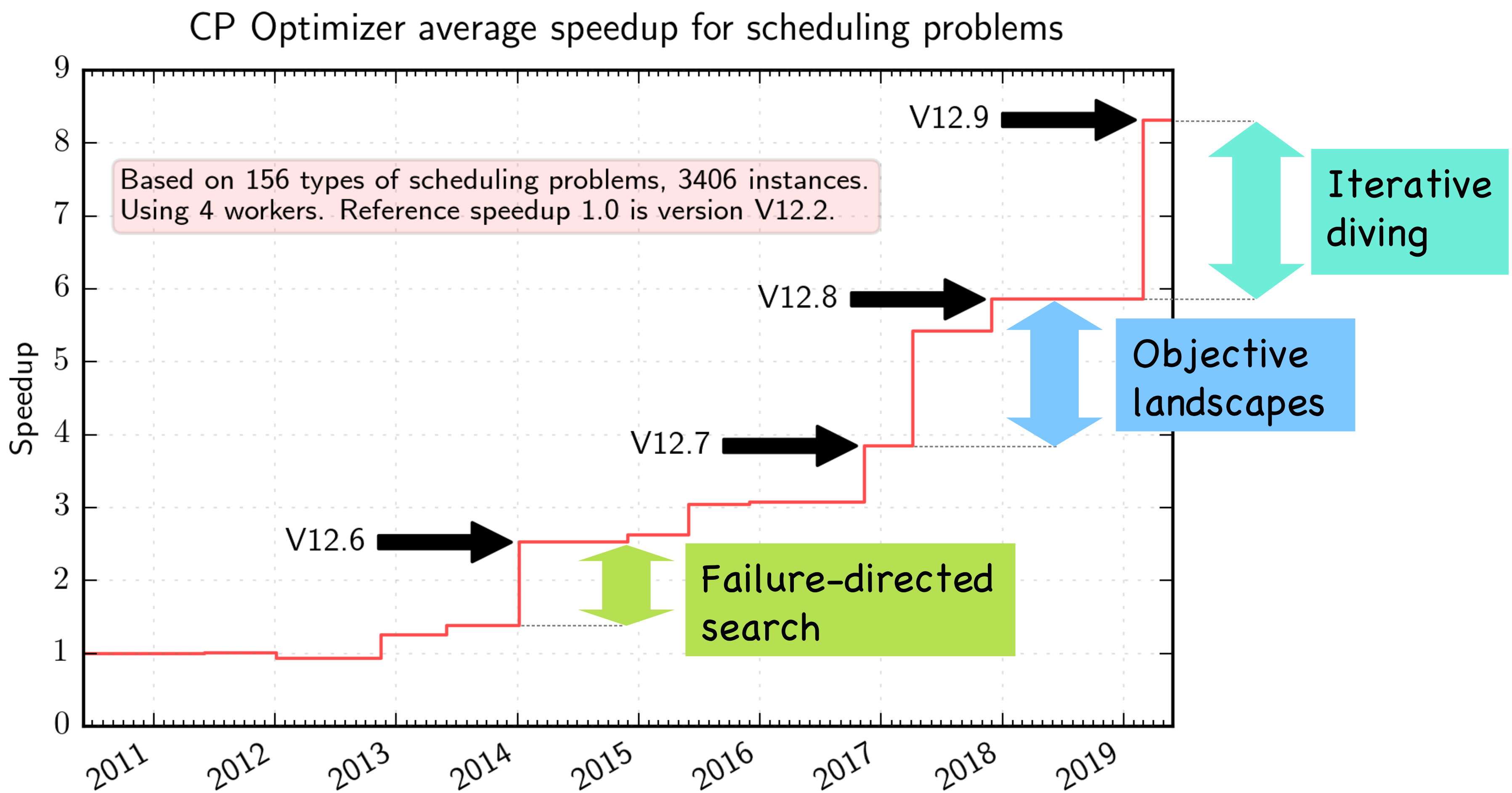
- The benchmark is mostly used to monitor the performance of the automatic search
- Though the search is **complete**, it is (still) not able to solve all problems to optimality 😊
- Each problem instance is run with a given time-limit on a given number of random seeds (search is **randomized**)
- Two versions of the search algorithm A and B are compared by computing a **speed-up ratio** that estimates how much faster the best algorithm (say A) finds a solution equivalent to the best solution found by the worst algorithm (here, B)
- Speed-up ratios are aggregated on the different problem instances to compute an **average speed-up ratio**

- Example

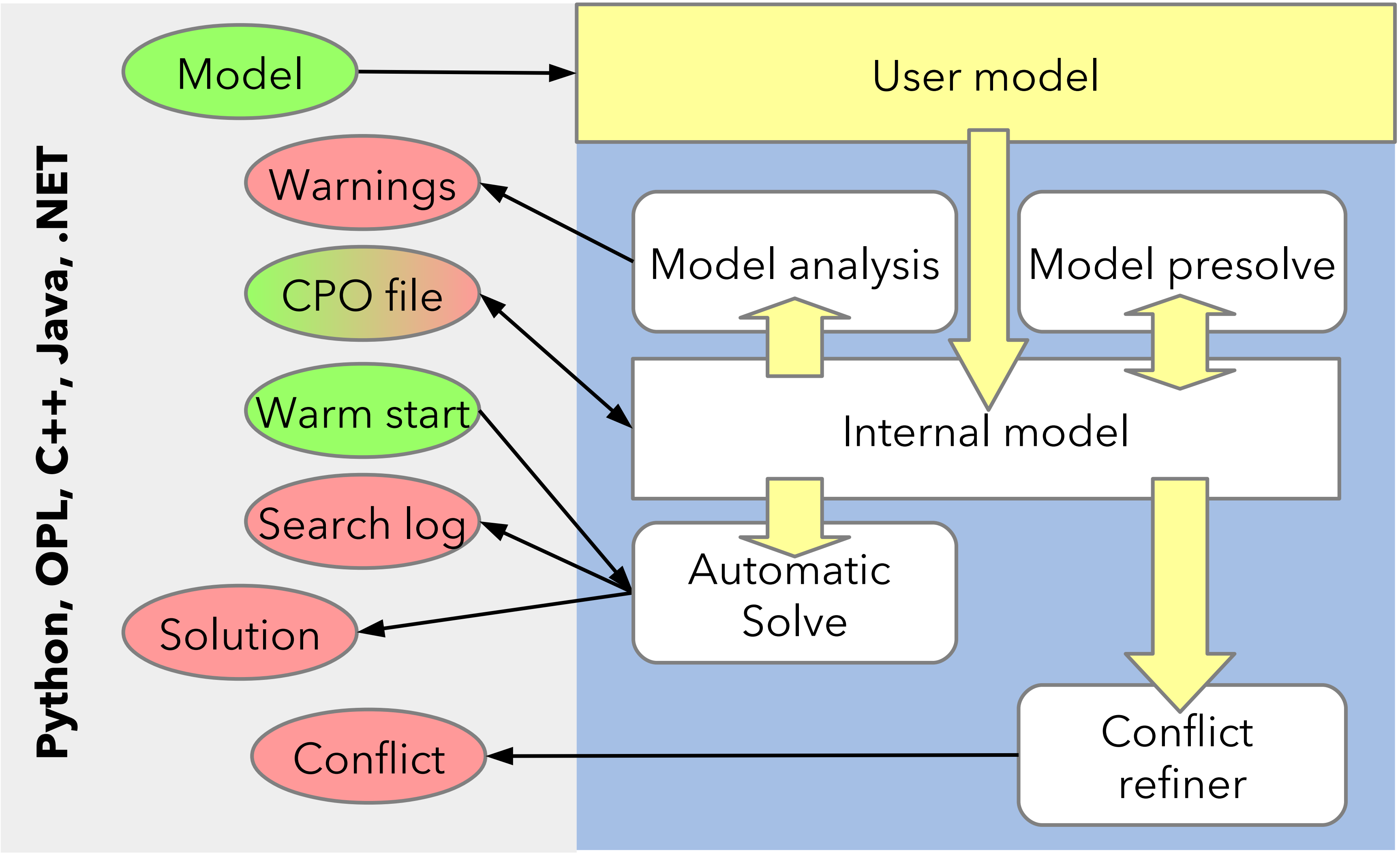

Problem: job-shop with earliness/tardiness
Instance: BFS_LT_2_20x10.cpo
V12.7
V12.7.1
All tests solutions (objective value / time)
Average A
Average B

CP Optimizer automatic search

- With 4 parallel workers, average speed-up of 42% between 12.8 and most recent version 12.9



CP Optimizer / CPLEX (MIP): a similar ecosystem



Conclusion

- A **mathematical modeling language** for combinatorial optimization problems that extends ILP (and classical CP) with some algebra on **intervals** and **functions** allowing compact and maintainable formulations for complex scheduling problems
- A continuously improving **automatic search algorithm** that is complete, anytime, efficient (e.g. competitive with problem-specific algorithms on classical problems) and scalable
- Recent review of CP Optimizer (modeling concepts, applications, examples, tools, performance,…) :
  - ***IBM ILOG CP Optimizer for scheduling***. Constraints (2018) 23:210–250. **http://ibm.biz/Constraints2018**

References

- P. Laborie, J. Rogerie. Reasoning with Conditional Time-Intervals. In: Proc. FLAIRS-2008.
- P. Laborie, J. Rogerie, P. Shaw, P. Vilím. Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources. In: Proc. FLAIRS-2009.

Modeling concepts

- P. Laborie. CP Optimizer for detailed scheduling illustrated on three problems. In: Proc. CPAIOR-2009.
- P. Laborie, B. Messaoudi. New Results for the GEO-CAPE Observation Scheduling Problem. In Proc. ICAPS-2017.
- P. Laborie. An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling. In Proc. CPAIOR-2018.

Examples

- P. Laborie, D. Godard. Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems. In: Proc. MISTA-2007.
- P. Vilím. Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources. In: Proc. CPAIOR-2011.
- P. Vilím, P. Laborie, P. Shaw. Failure-directed Search for Constraint-based Scheduling. In: Proc. CPAIOR-2015.
- P. Laborie, J. Rogerie. Temporal Linear Relaxation in IBM ILOG CP Optimizer. Journal of Scheduling, 19(4), 391–400 (2016).
- P. Laborie. Objective Landscapes for Constraint Programming. In Proc. CPAIOR-2018.

Search algorithm

- P. Laborie, J. Rogerie, P. Shaw, P. Vilím. IBM ILOG CP Optimizer for Scheduling. Constraints Journal, 23(2), 210-250 (2018). http://ibm.biz/Constraints2018

Overview