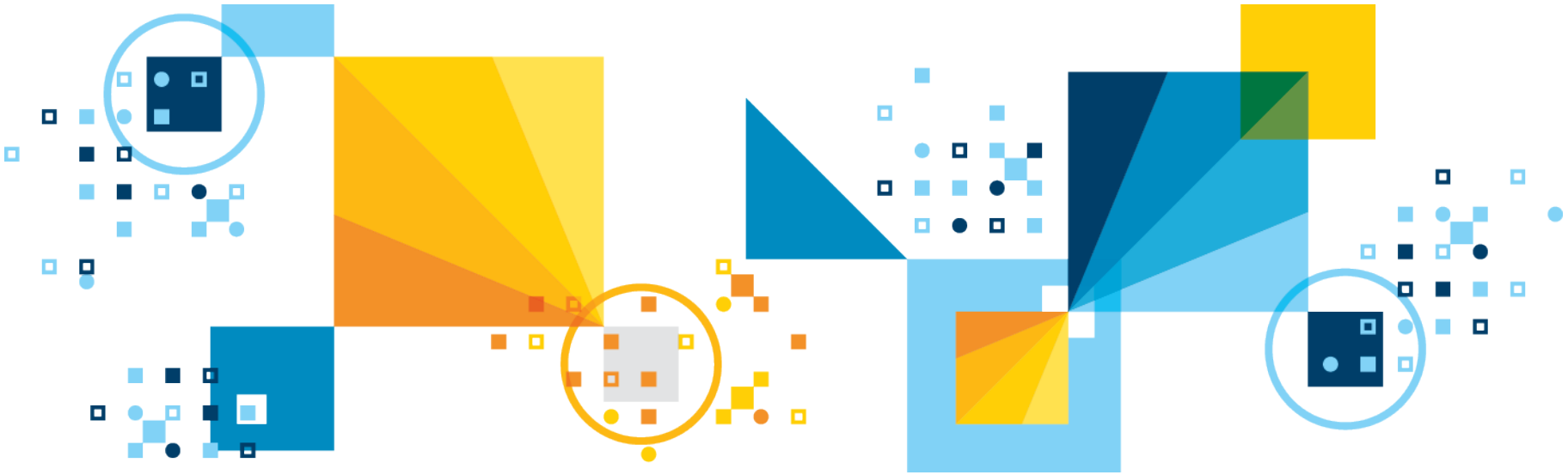
Philippe Laborie
02/04/2015

CP Optimizer updates



Agenda

- CP Optimizer overview
- Recent advances
  - Modeling
  - Tools
  - Automatic search
- Q&A

CP Optimizer overview

- A component of **IBM ILOG CPLEX Optimization Studio**
- A **Constraint Programming** engine for combinatorial problems (including detailed **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)
  - Java, .NET (wrapping of the C++ engine)

CP Optimizer overview

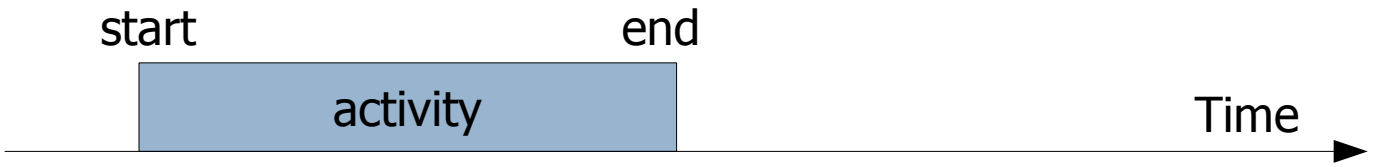
- Two main types of decision variables:
  - Integer variables
  - Interval variables

CP Optimizer overview : integer variables

Variables	Expressions	Constraints
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	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	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)

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 …)

CP Optimizer overview : interval variables (for scheduling)

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

  optional interval variable :

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

Absent interval

Interval of integers

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

CP Optimizer overview : 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, a recipe in a production order, an operation in a recipe
  - 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

CP Optimizer overview : interval variables (for scheduling)

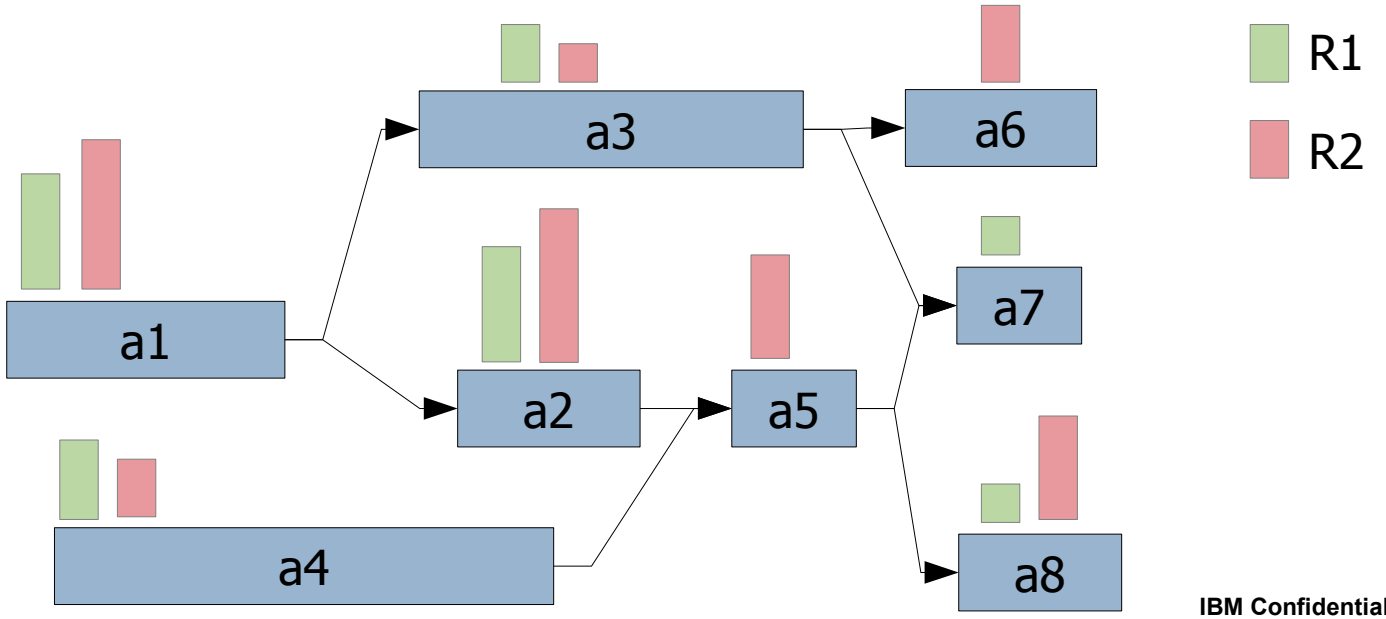
- An interval variable can be **optional** meaning that it is a decision to have it present or absent in a solution.

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

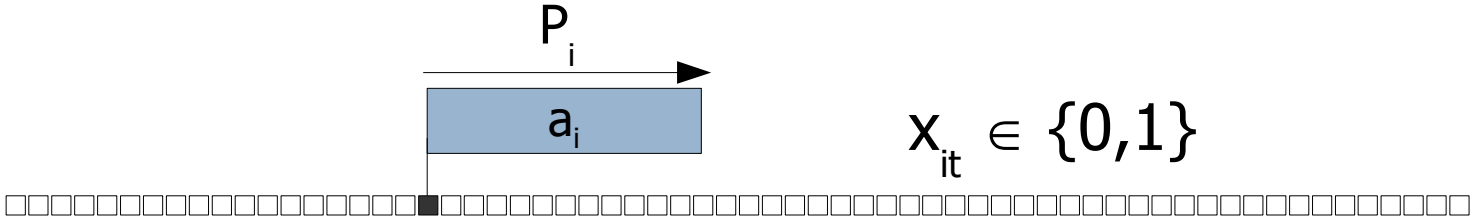
Example: Resource Constrained Project Scheduling Problem

- 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



Standard RCPSP (DT: Discrete Time)

$$\text{minimize} \sum_{t \in H} t x_{nt}$$

$$\sum_{t \in H} x_{it} = 1 \qquad \forall i \in \mathcal{A}$$

$$\sum_{t \in H} t x_{it} + P_i \leq \sum_{t \in H} t x_{jt} \qquad \forall (i,j) \in \mathcal{P}$$

$$\sum_{i \in A, t \leq \tau < t + P_i} Q_{ik} x_{it} \leq R_k \qquad \forall \tau \in H, \forall k \in \mathcal{R}$$

$$x_{it} \in \{0,1\} \qquad \forall i \in \mathcal{A}, \forall t \in H$$

Example: Resource Constrained Project Scheduling Problem

- Basic CP Optimizer model for RCPSP:

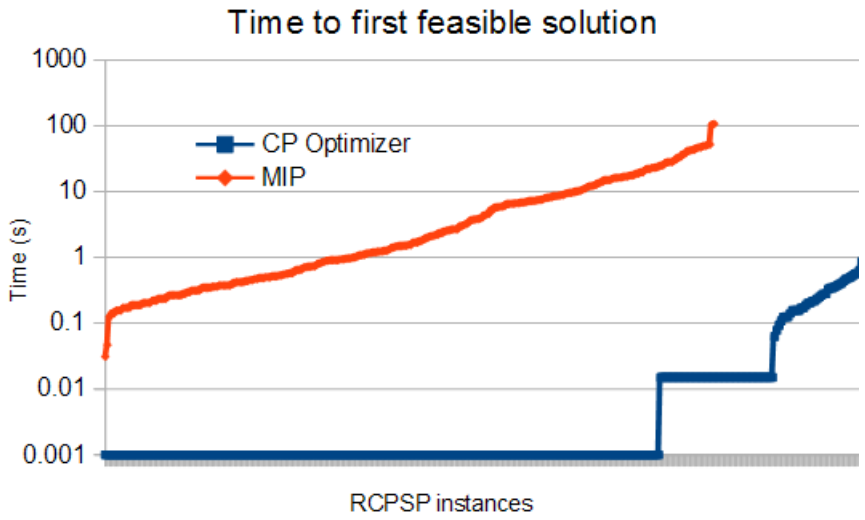
```
dvar interval a[i in Tasks] size i.pt;

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>]);
}
```

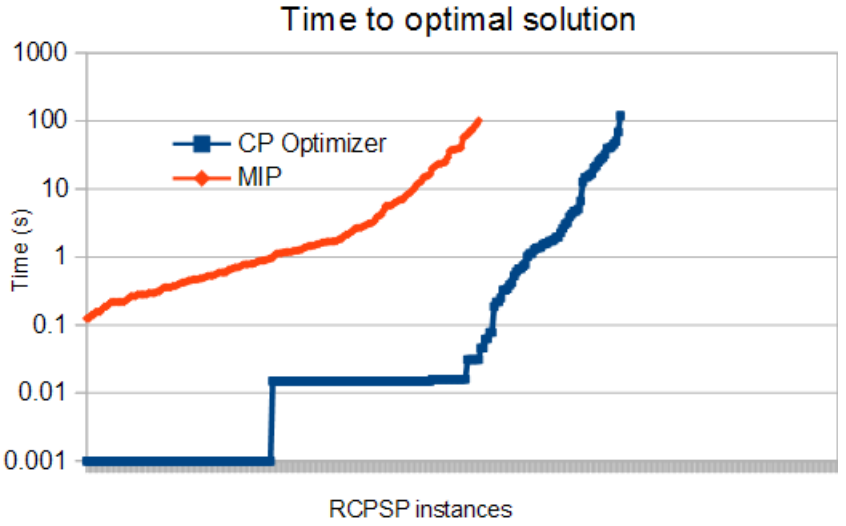
Example: Resource Constrained Project Scheduling Problem

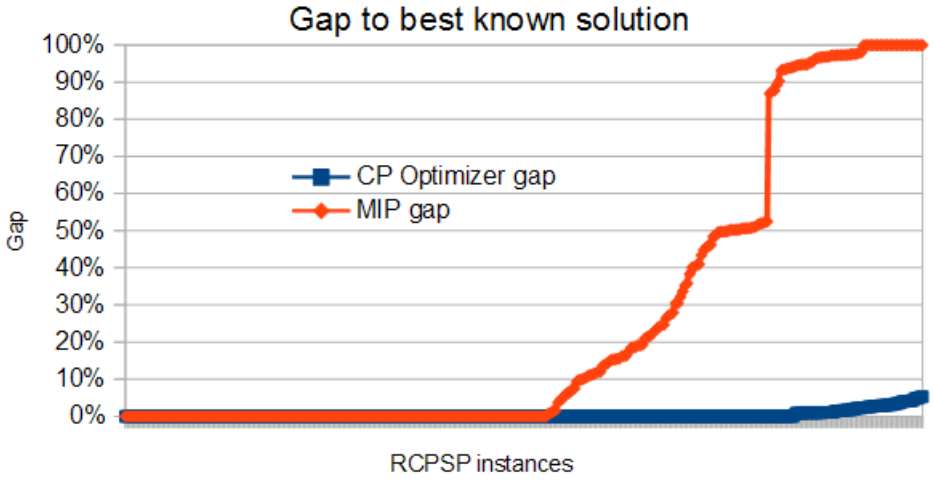
- Comparison of this time-indexed MIP formulation against the CP Optimizer model on a set of:
  - 300 classical **small** RCPSP instances (30-120 tasks) +
  - 40 slightly **more realistic** larger ones (900 tasks)
  - time-limit: 2mn, 4 threads
- Note: industrial scheduling problems are often much **larger**, 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

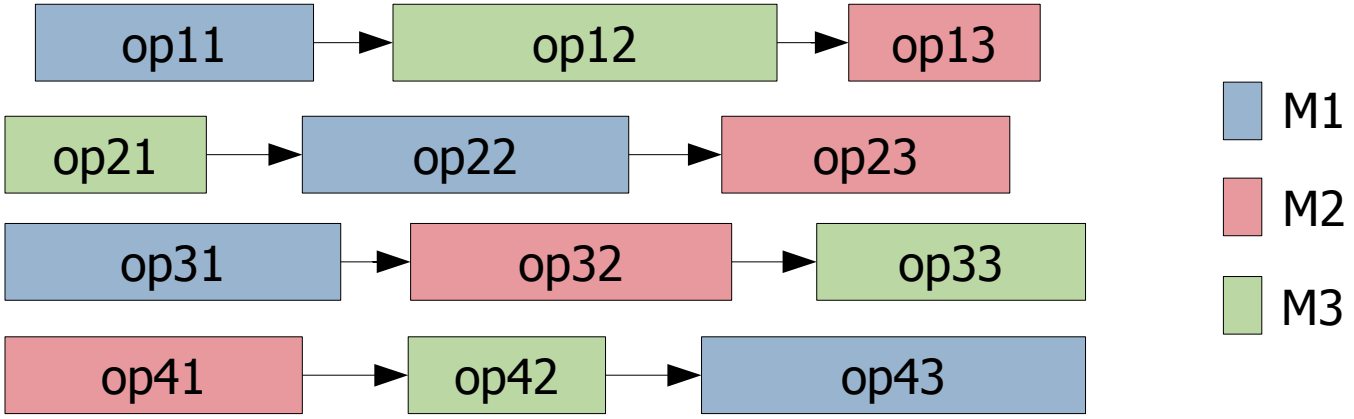






Example : Job-shop Scheduling Problem

- Example: Job-shop Scheduling Problem



- Minimization of makespan

Example : Job-shop Scheduling Problem

- CP Optimizer model for Job-shop:

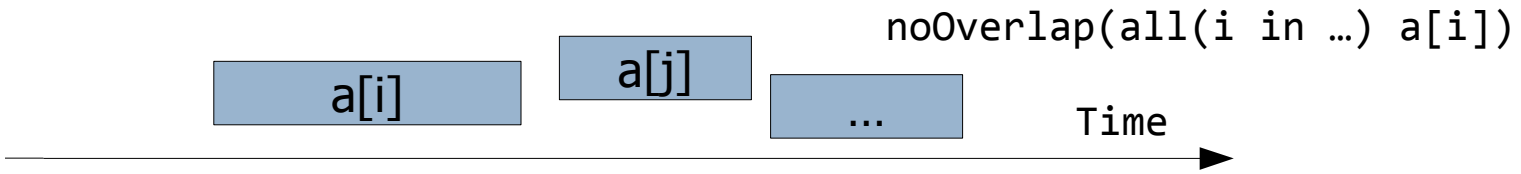
```
dvar interval op[j in Jobs][p in Pos] size Ops[j][p].pt;
dvar sequence mchs[m in Mchs] in
  all(j in Jobs, p in Pos : Ops[j][p].mch == m) op[j][p];

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:

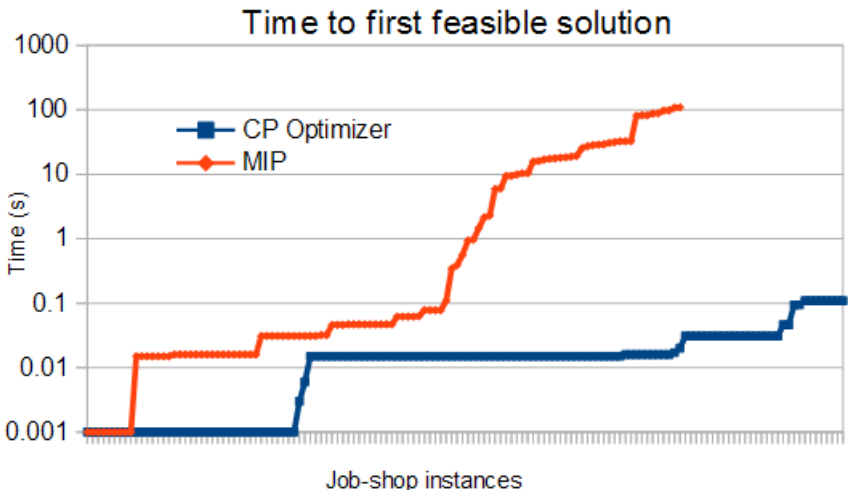


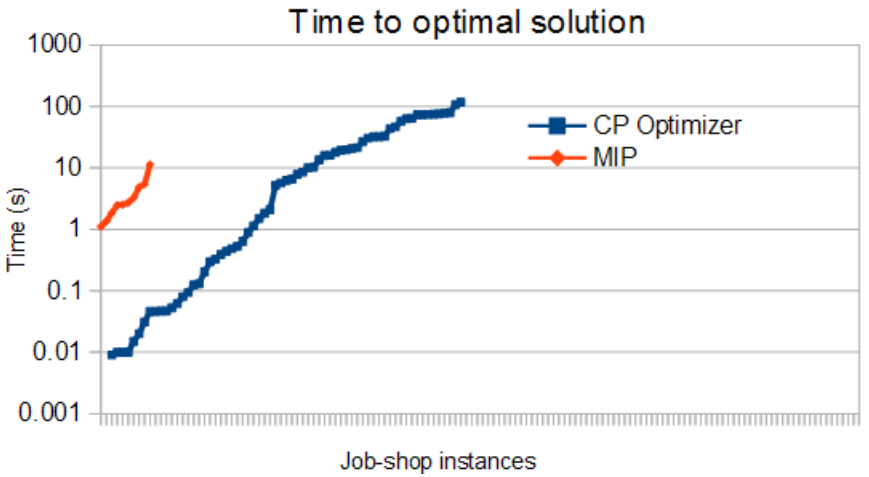
Example : Job-shop Scheduling Problem

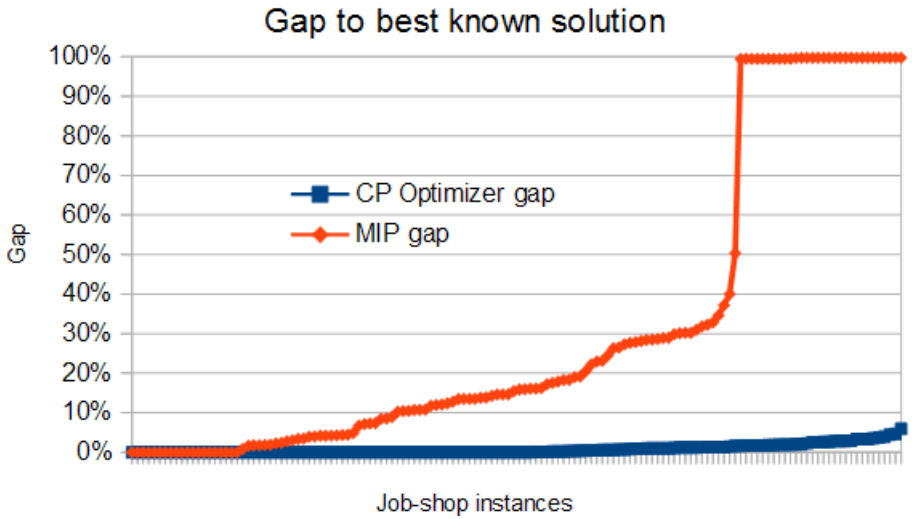
- Comparison of the CP Optimizer model vs a disjunctive MIP formulation on a set of 140 classical Job-shop instances (50-2000 tasks), time-limit: 2mn, 4 threads

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]);
}
```

Agenda

- CP Optimizer overview
- Recent advances
  - Modeling
  - Tools
  - Automatic search
- Q&A

Agenda

- CP Optimizer overview
- Recent advances
  - Modeling
  - Tools
  - Automatic search
- Q&A

- Lexicographical objectives (12.3)
- OverlapLength expressions (12.2)
- SameSequence constraints (12.6)

- Starting points (old but so useful)
- Conflict refiner (12.5)
- I/O Format (12.6.1)
- Strong annotations (12.6)

- Presolve (12.5)
- Failure-Directed Search (12.6)

Modeling : Lexicographical objectives

- Hierarchical objectives are common in applications

```
dexpr int crit1 = …;
dexpr int crit1 = …;
minimize staticLex(crit1, crit2, …);
```

- During the search the objective cut eliminates the dominated objective vectors

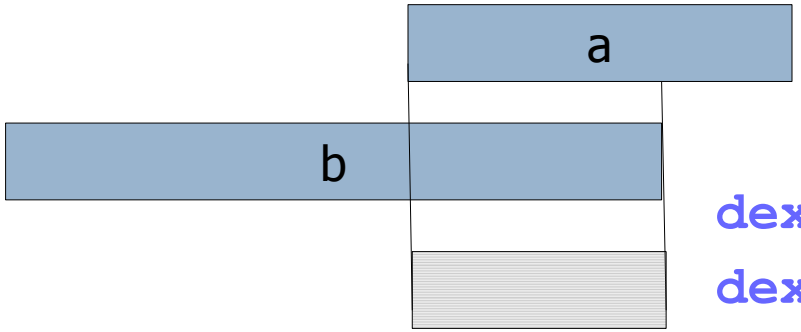
Modeling : Lexicographical objectives

- Example of search log:

```
! ------------------------------------------------------------------------
! Minimization problem - 103 variables, 341 constraints, 1 phase
! Initial process time : 0.01s (0.01s extraction + 0.00s propagation)
!  . Log search space  : 319.3 (before), 319.3 (after)
!  . Memory usage      : 1.7 MB (before), 1.7 MB (after)
! Using sequential search.
! ------------------------------------------------------------------------
!          Best Branches  Non-fixed            Branch decision
*            53       38 0.01s                  0  = transitionCost(15)
. Multi-criterion values are 53; 11
*            48    3,581 0.09s                  0  = transitionCost(15)
...
. Multi-criterion values are 28; 14
*            26   33,375 0.88s                  1  = customerOfTruck(15)
. Multi-criterion values are 26; 13
! ------------------------------------------------------------------------
! Search terminated by limit, 17 solutions found.
! Best objective         : 26; 13
! Number of branches     : 745,384
! Number of fails        : 359,944
! Total memory usage     : 3.0 MB (2.4 MB CP Optimizer + 0.6 MB Concert)
! Time spent in solve    : 20.01s (19.99s engine + 0.01s extraction)
! Search speed (br. / s) : 37,271.1
! ------------------------------------------------------------------------
```

Modeling : OverlapLength expressions

- An expression to return the length of the overlap between 2 interval variables



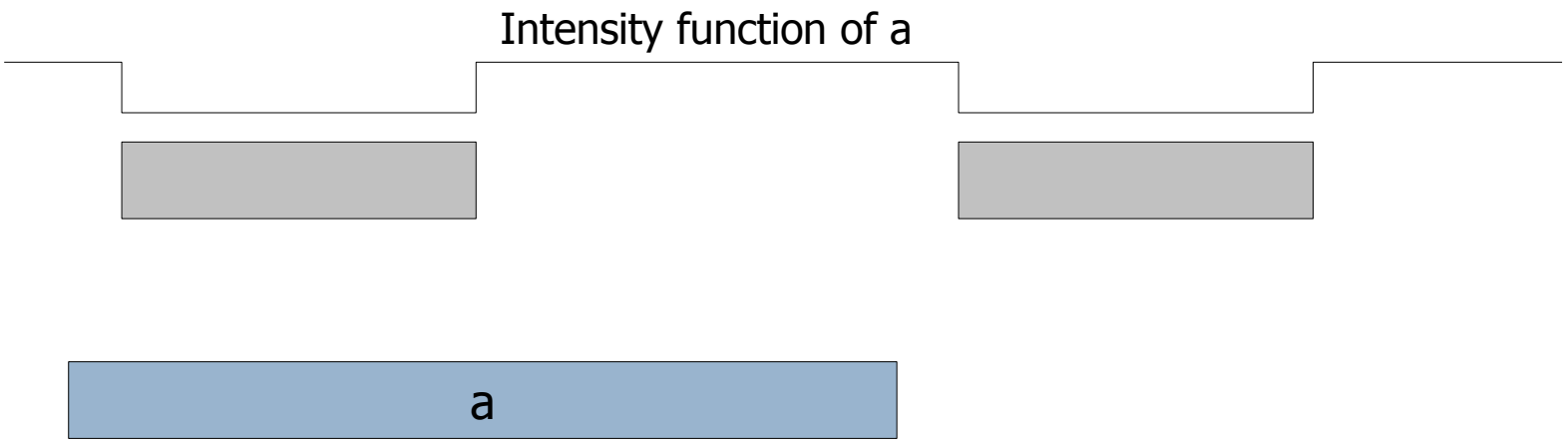

```
dexpr int overlapLength(a,b)
dexpr int overlapLength(a,t1,t2)
```

- Use cases:
  - Time buckets with limited energy:

$$\sum_i q_i * \text{overlapLength}(a_i, t_1, t_2) \leq W$$

Modeling : OverlapLength expressions

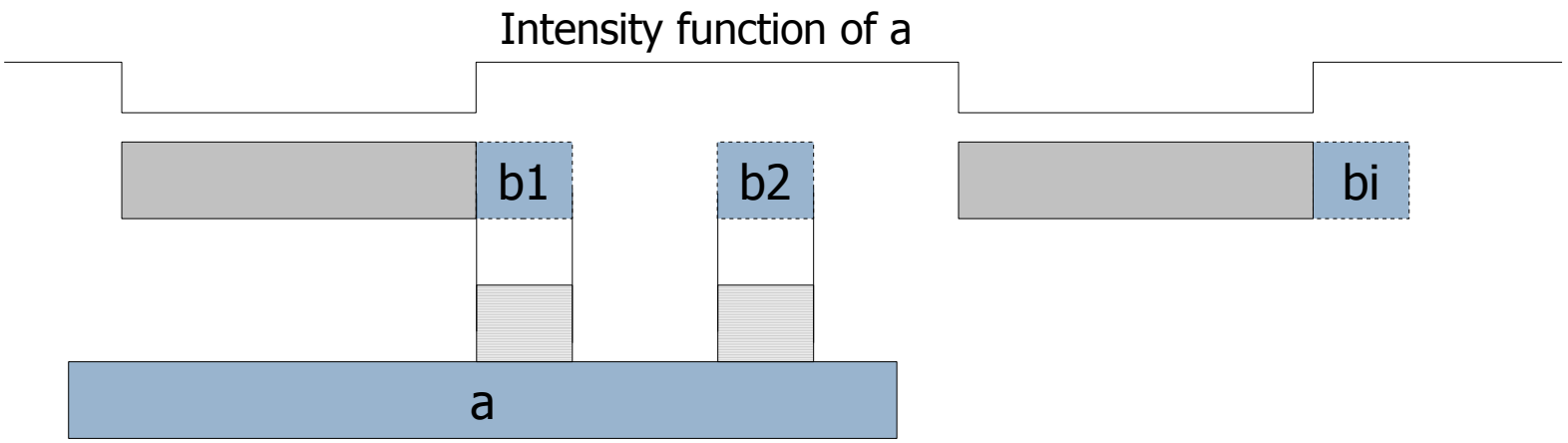
- Use cases:
  - Flexible breaks on a resource



Modeling : OverlapLength expressions

- Use cases:
  - Flexible breaks on a resource:

$$\texttt{size(a)} = \texttt{sizeMin} + \sum_i \texttt{overlapLength(a,b}_i\texttt{)}$$

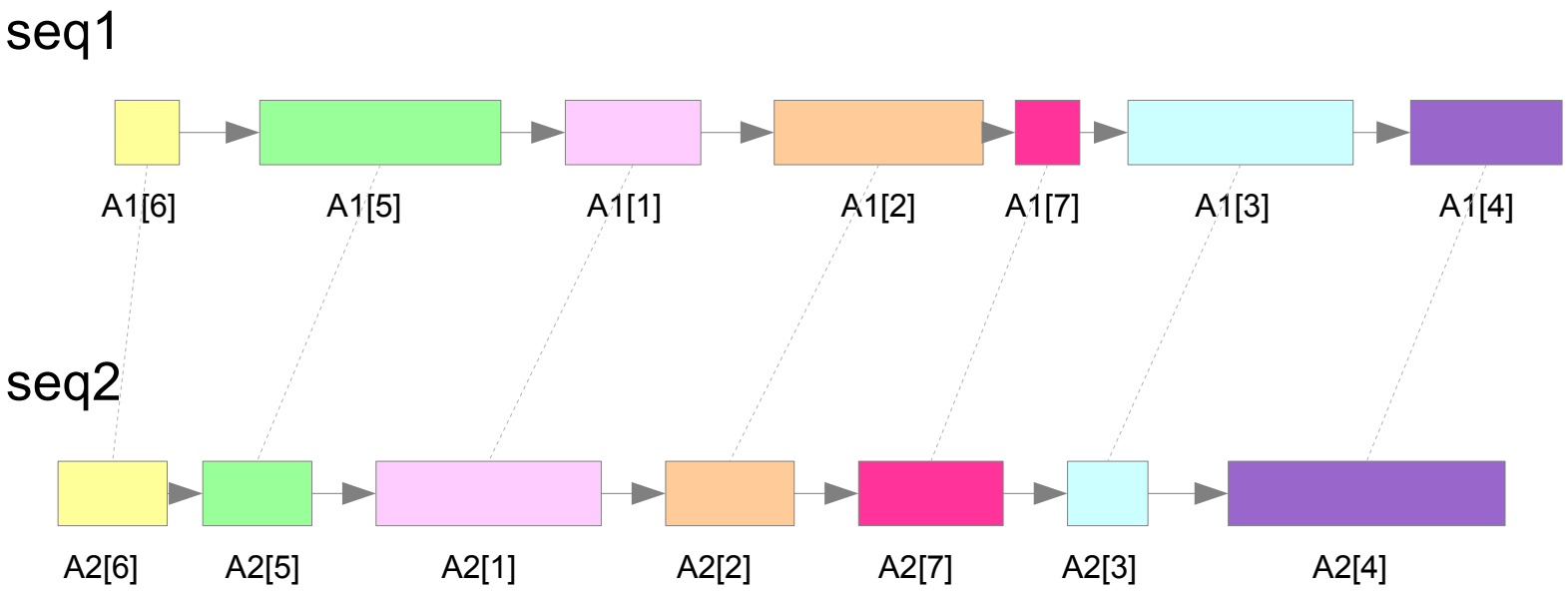


$$\texttt{maximize } \sum_i \texttt{ presenceOf(b}_i\texttt{)}$$

Modeling : SameSequence constraints

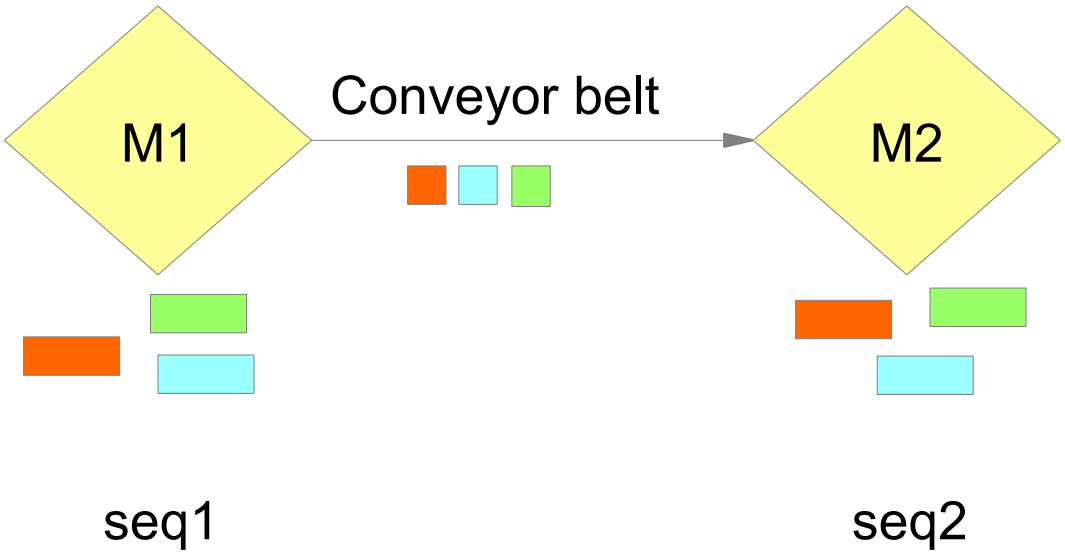
- A new constraint to (partially) map two sequence variables

```
sameSequence(seq1,seq2)
```



Modeling : SameSequence constraints

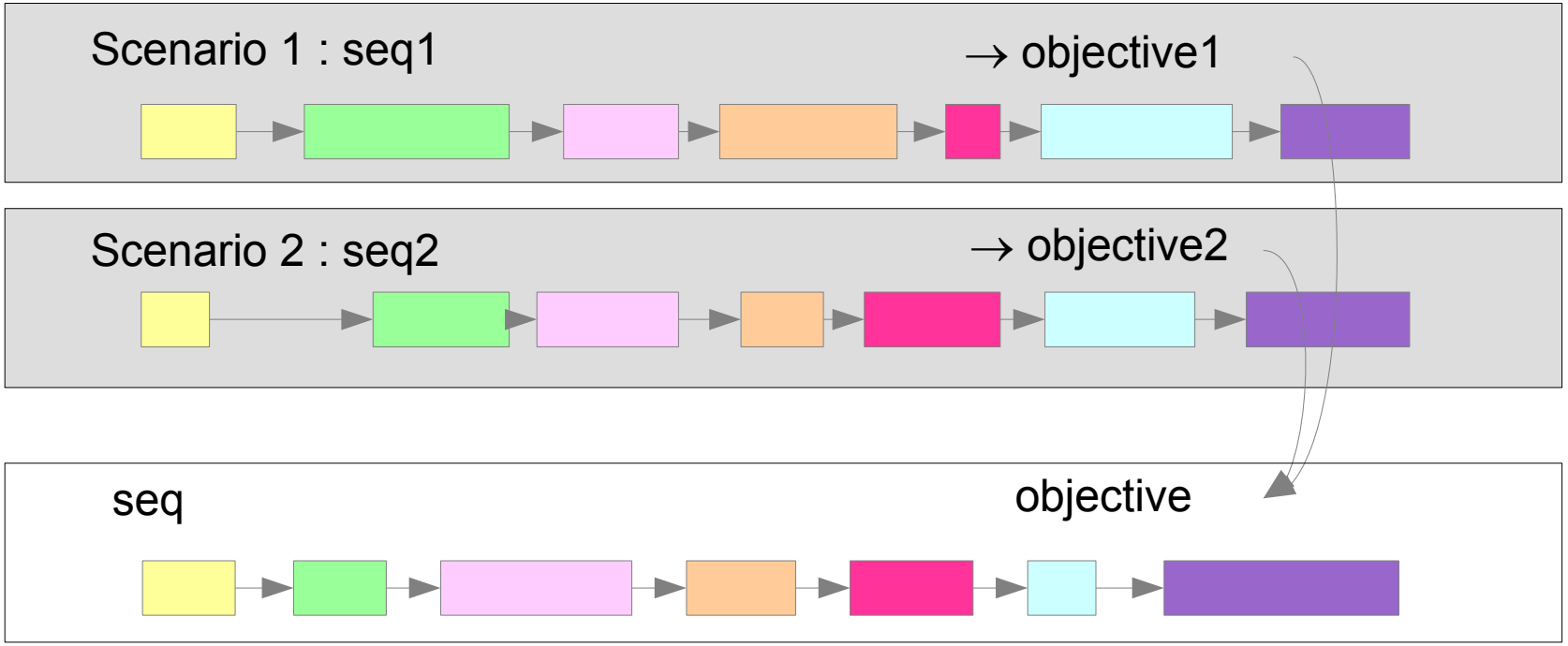
- Use case:
  - No-bypass constraint




```
sameSequence(seq1,seq2)
```

Modeling : SameSequence constraints

- Use case:
  - Stochastic scheduling: sequences as 1st stage variables in 2 stage stochastic programming



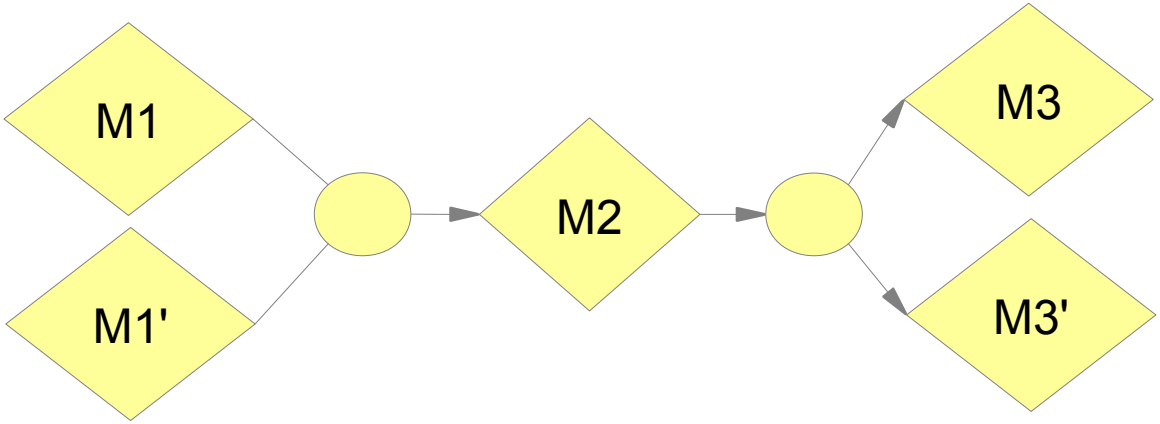
sameSequence(seq,seq$_i$)

Modeling : SameSequence constraints

- A variant ensuring a partial mapping only on the pairs of present interval variables

```
sameCommonSequence(seq1,seq2)
```

- Use cases: No-bypass constraints with junctions




```
sameCommonSequence(seq1, seq2)
sameCommonSequence(seq1',seq2)
sameCommonSequence(seq2, seq3)
sameCommonSequence(seq2, seq3')
```

Tools: Starting points

- Objective: Start search from a known (possibly incomplete) solution given by the user (warm start) in order to further improve it or to help to guide the engine towards a first feasible solution
- API: `IloCP::setStartingPoint(IloSolution warmstart)`
- Use cases:
  - Restart an interrupted search with the current incumbent
  - Start from an initial solution found by an available heuristic
  - Goal programming for multi-objective problems
  - When finding an initial solution is hard, solve an initial problem that maximizes constraint satisfaction and start from its solution
  - Successively solving similar problems (e.g. dynamic scheduling)
  - Hierarchical problem solving (e.g. planning → scheduling)

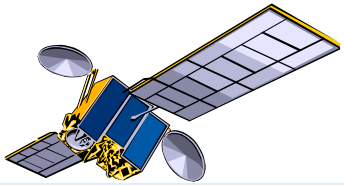
Tools: Conflict refiner

- Objective: Identify a reason for an inconsistency by providing a minimal infeasible subset of constraints for an infeasible model
- API: `IloCP::refineConflict()`
  `IloCP::writeConflict(ostream& out)`
- Use cases:
  - Model debugging (errors in model)
  - Data debugging (inconsistent data)
  - The model and data are correct, but the associated data represents a real-world conflict in the system being modeled
  - You create an infeasible model to test properties of (or extract information about) a similar model

Conflict Refiner example: satellite scheduling problem

- Satellite Control Network scheduling problem [1]
- *n* communication requests for Earth orbiting satellites must be scheduled on a total of 32 antennas spread across 13 ground-based tracking stations
- In the instances, *n* ranges from to 400 to 1300

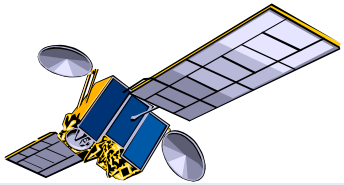
[1] Kramer & al.: Understanding Performance Trade-offs in Algorithms for Solving Oversubscribed Scheduling.

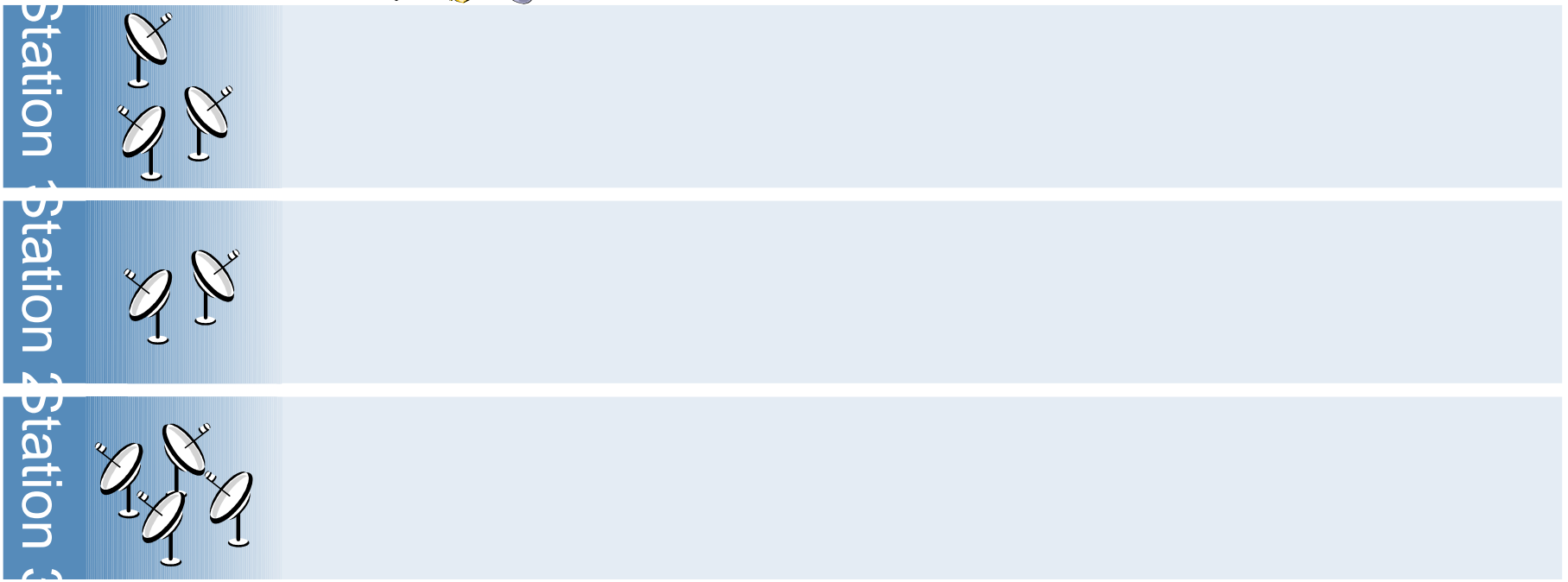


Station

Station

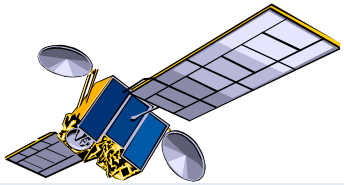
Station

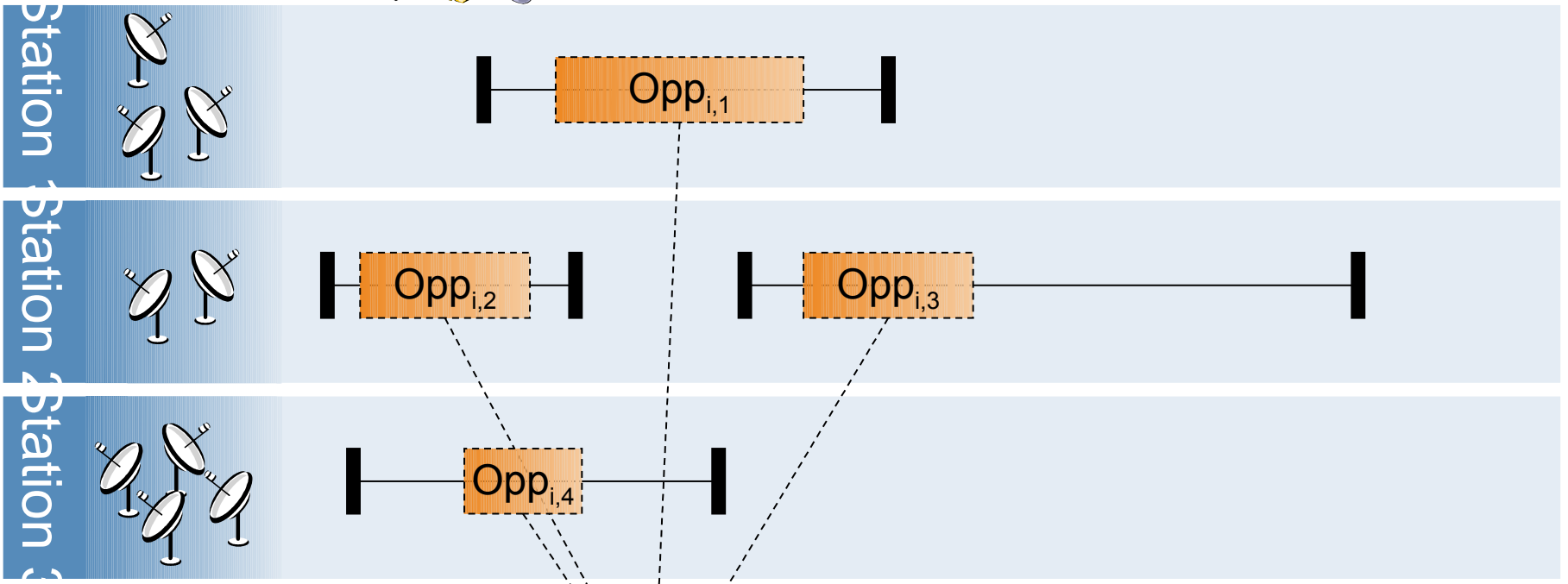




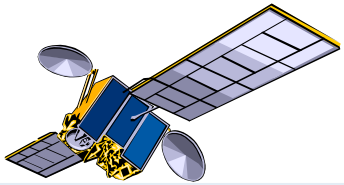
Task$_i$

Communication requests

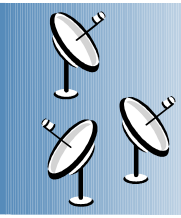




Alternative assignments to stations × time windows (opportunities)



Station 1



$Opp_{i,1}$

Station 2

$Opp_{i,2}$

$Opp_{i,3}$

Station 3

$Opp_{i,4}$

$Task_i$

Selected opportunity will use 1 antenna for communication with the satellite

Conflict Refiner example: model 1

```
using CP;

tuple Station {
  string name; // Ground station name
  int id;      // Ground station identifier
  int cap;     // Number of available antennas
}

tuple Opportunity {
  string task; // Task
  int station; // Ground station
  int smin;    // Start of visibility window of opportunity
  int dur;     // Task duration in this opportunity
  int emax;    // End of visibility window of opportunity
}

{Station} Stations = ...;
{Opportunity} Opportunities = ...;
{string} Tasks = { o.task | o in Opportunities };

dvar interval task[t in Tasks];
dvar interval opp[o in Opportunities] in o.smin..o.emax size o.dur;


subject to {
  forall(t in Tasks)
    opportunitySelection: alternative(task[t], all(o in Opportunities: o.task==t) opp[o]);
  forall(s in Stations)
    numberOfAntennas: sum(o in Opportunities: o.station==s.id) pulse(opp[o],1) <= s.cap;
}
```

Conflict Refiner example: running model 1

```
! ----------------------------------------------------------------------
! Satisfiability problem - 2,980 variables, 851 constraints
! Workers               = 2
! TimeLimit             = 30
! Problem found infeasible at the root node
! ----------------------------------------------------------------------
  ...
! ----------------------------------------------------------------------
! Conflict refining - 851 constraints
! ----------------------------------------------------------------------
!   Iteration      Number of constraints
*           1                        851
  ...
*          11                          3
*          12                          1
*          13                          1
! Conflict refining terminated
! ----------------------------------------------------------------------
! Conflict status           : Terminated normally, conflict found
! Conflict size             : 1 constraint
! Number of iterations      : 13
! Total memory usage        : 10.6 MB
! Conflict computation time : 0.04s
! ----------------------------------------------------------------------
```

Conflict Refiner example: running model 1

- Conflict:

Line	In conflict	Element (1)
26	Yes	opportunitySelection["373A"]

- Opportunities for task “373A”:

```
<"373A",2,1191,23,1241>,
<"373A",3,1191,23,1241>,
<"373A",11,1191,23,1241>,
<"373A",13,1191,23,1241>,
<"373A",5,1191,23,1241>,
<"373A",7,1191,23,1241>,
<"373A",8,1191,23,1241>,
```

Conflict Refiner example: model 1

```
1  using CP;
2
3  tuple Station {
4    string name; // Ground station name
5    int id;      // Ground station identifier
6    int cap;     // Number of available antennas
7  }
8
9  tuple Opportunity {
10   string task; // Task
11   int station; // Ground station
12   int smin;    // Start of visibility window of opportunity
13   int dur;     // Task duration in this opportunity
14   int emax;    // End of visibility window of opportunity
15 }
16
17 {Station} Stations = ...;
18 {Opportunity} Opportunities = ...;
19 {string} Tasks = { o.task | o in Opportunities };
20
21 dvar interval task[t in Tasks];
22 dvar interval opp[o in Opportunities] in o.smin..o.emax size o.dur;
23
24
25 subject to {
26   forall(t in Tasks)
27     opportunitySelection: alternative(task[t], all(o in Opportunities: o.task==t) opp[o]);
28   forall(s in Stations)
29     numberOfAntennas: sum(o in Opportunities: o.station==s.id) pulse(opp[o],1) <= s.cap;
30 }
```

Conflict Refiner example: model 2

```
1  using CP;
2
3  tuple Station {
4    string name; // Ground station name
5    int id;      // Ground station identifier
6    int cap;     // Number of available antennas
7  }
8
9  tuple Opportunity {
10   string task; // Task
11   int station; // Ground station
12   int smin;    // Start of visibility window of opportunity
13   int dur;     // Task duration in this opportunity
14   int emax;    // End of visibility window of opportunity
15 }
16
17 {Station} Stations = ...;
18 {Opportunity} Opportunities = ...;
19 {string} Tasks = { o.task | o in Opportunities };
20
21 dvar interval task[t in Tasks];
22 dvar interval opp[o in Opportunities] optional in o.smin..o.emax size o.dur;
23
24
25 subject to {
26   forall(t in Tasks)
27     opportunitySelection: alternative(task[t], all(o in Opportunities: o.task==t) opp[o]);
28   forall(s in Stations)
29     numberOfAntennas: sum(o in Opportunities: o.station==s.id) pulse(opp[o],1) <= s.cap;
30 }
```

Conflict Refiner example: running model 2

```
! ----------------------------------------------------------------------
! Satisfiability problem - 2,980 variables, 851 constraints
! Problem found infeasible at the root node
! ----------------------------------------------------------------------
  ...
! ----------------------------------------------------------------------
! Conflict refining - 851 constraints
! ----------------------------------------------------------------------
!   Iteration      Number of constraints
*           1                        851
*           2                        426
 ...
*          58                          5
*          59                          5
! Conflict refining terminated
! ----------------------------------------------------------------------
! Conflict status           : Terminated normally, conflict found
! Conflict size             : 5 constraints
! Number of iterations      : 59
! Total memory usage        : 13.3 MB
! Conflict computation time : 0.51s
! ----------------------------------------------------------------------
```

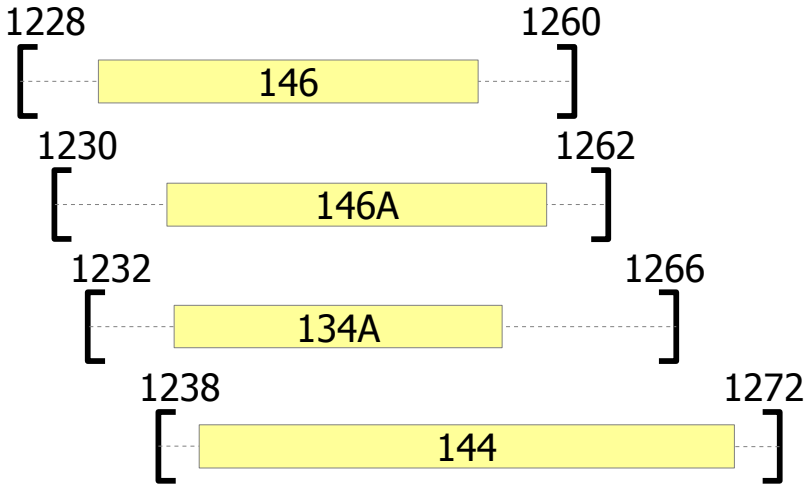
Conflict Refiner example: running model 2

- Conflict:

Line	In conflict	Element (5)
26	Yes	opportunitySelection["134A"]
26	Yes	opportunitySelection["144"]
26	Yes	opportunitySelection["146"]
26	Yes	opportunitySelection["146A"]
28	Yes	numberOfAntennas[<"LION",6,3>]

- There is not enough antennas to accommodate all 4 tasks on their time-window on ground station "LION" (3 antennas):

```
<"134A",6,1232,19,1266>
<"144", 6,1238,31,1272>
<"146", 6,1228,22,1260>
<"146A",6,1230,22,1262>
```



Conflict Refiner example: model 2

```
1  using CP;
2
3  tuple Station {
4    string name; // Ground station name
5    int id;      // Ground station identifier
6    int cap;     // Number of available antennas
7  }
8
9  tuple Opportunity {
10   string task; // Task
11   int station; // Ground station
12   int smin;    // Start of visibility window of opportunity
13   int dur;     // Task duration in this opportunity
14   int emax;    // End of visibility window of opportunity
15 }
16
17 {Station} Stations = ...;
18 {Opportunity} Opportunities = ...;
19 {string} Tasks = { o.task | o in Opportunities };
20
21 dvar interval task[t in Tasks];
22 dvar interval opp[o in Opportunities] optional in o.smin..o.emax size o.dur;
23
24
25 subject to {
26   forall(t in Tasks)
27     opportunitySelection: alternative(task[t], all(o in Opportunities: o.task==t) opp[o]);
28   forall(s in Stations)
29     numberOfAntennas: sum(o in Opportunities: o.station==s.id) pulse(opp[o],1) <= s.cap;
30 }
```

Conflict Refiner example: model 3

```
1  using CP;
2
3  tuple Station {
4    string name; // Ground station name
5    int id;      // Ground station identifier
6    int cap;     // Number of available antennas
7  }
8
9  tuple Opportunity {
10   string task; // Task
11   int station; // Ground station
12   int smin;    // Start of visibility window of opportunity
13   int dur;     // Task duration in this opportunity
14   int emax;    // End of visibility window of opportunity
15 }
16
17 {Station} Stations = ...;
18 {Opportunity} Opportunities = ...;
19 {string} Tasks = { o.task | o in Opportunities };
20
21 dvar interval task[t in Tasks] optional;
22 dvar interval opp[o in Opportunities] optional in o.smin..o.emax size o.dur;
23
24 maximize sum(t in Tasks) presenceOf(task[t]);
25 subject to {
26   forall(t in Tasks)
27     opportunitySelection: alternative(task[t], all(o in Opportunities: o.task==t) opp[o]);
28   forall(s in Stations)
29     numberOfAntennas: sum(o in Opportunities: o.station==s.id) pulse(opp[o],1) <= s.cap;
30 }
```

Tools: I/O Format

- API: `IloCP::exportModel(const char* name)`
  `IloCP::importModel(const char* name)`
- Can be used at top level or during search to dump a partial state of the engine
- Human-readable input/output file format

```
// ------------------------------------------------------------------------
// 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)");
...
```

Tools: Strong annotations

- Strong(x1, x2 … xn): automatically generate an allowedAssignment (a.k.a. table) constraint over variables x1, x2 … xn
- Benefits:
  - Reinforce the model propagation between variables where user know that the domain reduction is weak
  - Avoid the enumeration of tuple sets for creating allowedAssignments
  - Create stronger allowedAllowed assignments than those obtained by hand
- API:
  - In C++: `IloStrong(IloIntVarArray x)`
  - In OPL: `strong(all …)`

Tools: Strong annotations

- Basic generation
  - Enumerate solutions by performing a search on the variables of the strong using domain reduction with the whole model
  - Add an allowedAssignment over the variables with the tupleSet of the generated solutions and remove constraints over these variables that have become redundant
- Automatic generation
  - After a given number of node explored for solutions, decide if it worths continuing generating
  - Criteria 1 : the expected number of solutions (too many solutions gives heavy allowedAssignments).
  - Criteria 2 : measure the impact of the constraint by looking at the search space reduction (a weak constraint may not be beneficial)

Tools: Strong annotations : Football tournament example

```
IloIntTupleSet gha(env, 3);
IloIntArray tuple(env, 3);
for (IloInt i = 0; i < n; i++) {
  tuple[0] = i;
  for (IloInt j = 0; j < n; j++) {
    if (i != j) {
      tuple[1] = j;
      tuple[2] = Game(i, j, n);
      gha.add(tuple);
    }
  }
}

for (IloInt i = 0; i < nbWeeks; i++) {
  for (IloInt j = 0; j < nbGamesPerWeek; j++) {
    IloIntVarArray vars(env);
    vars.add(home[i][j]);
    vars.add(away[i][j]);
    vars.add(games[i][j]);
    model.add(IloAllowedAssignments(env, vars, gha));
  }
}
```

```
!          Best Branches  Non-fixed
*            56     2688 0.53s
*            52     7402 0.99s
*            48    19418 2.38s
*            46     102k 11.87s
*            44     109k 12.81s
*            42     195k 20.22s
*            40     218k 22.43s
*            38     239k 24.47s
```

```
for (IloInt i = 0; i < nbWeeks; i++) {
  for (IloInt j = 0; j < nbGamesPerWeek; j++) {
    model.add(home[i][j] != away[i][j]);
    model.add(games[i][j] == Game(home[i][j], away[i][j], n));
    IloIntVarArray vars(env);
    vars.add(home[i][j]);
    vars.add(away[i][j]);
    vars.add(games[i][j]);
    model.add(IloStrong(env, vars));
  }
}
```

```
!          Best Branches  Non-fixed    W       Branch
  decision
*            50      520 0.44s        6            -
*            46     2468 0.65s        1            -
*            38     4304 0.88s        1            -
*            36    25689 2.98s        1            -
*            34    86973 8.15s        1            -
```

The original tuple set has 90 tuples, the generated ones have between 5 and 66 tuples.

Automatic Search: 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

Automatic Search: 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

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

  $x == 4 \rightarrow 21y \leq 45$
- Variable merge

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

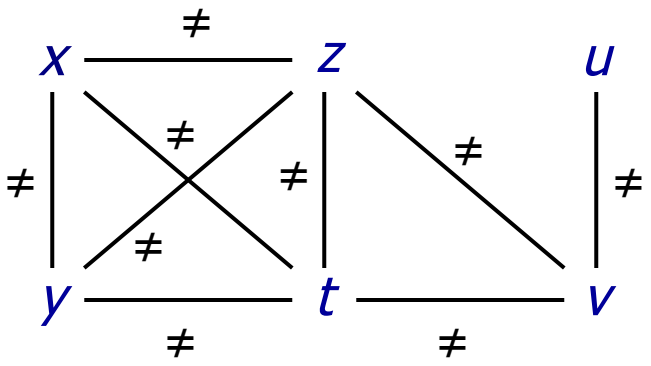
Automatic Search: Examples of Model Presolve

- 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

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

Automatic Search: 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

Automatic Search: Failure-Directed Search

- An additional technique recently added in the automatic search portfolio for scheduling problems
- FDS:
  - assumes that there is no better solution than the current solution or that such an improving solution is very hard to find
  - is based on a systematic exploration of the search tree using a first-fail approach
  - uses restarts, no-good learning and some sort form impact measurements
  - works on choice points of the form:
    - presence/absent of interval
    - Start/end of interval <= date value
- FDS is used as a “plan B” strategy used once a less systematic « plan A » strategy (LNS) is not able to improve the current solution anymore

Automatic Search

- Search algorithm is **Complete**
- 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 old 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 2015

Some recent 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

Automatic Search

Average speed-up between V12.5 and V12.6.1 on a couple of models that may ring a bell to some of you:

	Speed-up ratio 12.6.1 / 12.5 on bench platform	Geometrical mean
DetailedProjectScheduling_1	1,01	7,55
DetailedProjectScheduling_2	20,82 *	
DetailedProjectScheduling_3	16,86 *	
DetailedProjectScheduling_4	9,16	
YogurtScheduling_1	1,65	2,29
YogurtScheduling_2	1,21	
YogurtScheduling_3	0,8	
YogurtScheduling_4	0,83	
YogurtScheduling_5	4,64	
YogurtScheduling_6	4,46	
YogurtScheduling_7	3,83	
YogurtScheduling_8	3,88	
YogurtScheduling_9	3,14	
YogurtScheduling_10	3,11	

Comparison CPLEX / CP Optimizer

- 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 papers related with CP Optimizer

P. Vilim, P. Laborie and P. Shaw. **“Failure-directed Search for Constraint-based Scheduling”**. CP-AI-OR 2015.

P. Laborie. **“An Optimal Iterative Algorithm for Extracting MUCs in a Black-box Constraint Network”**. ECAI 2014.

P. Laborie and J. Rogerie. **“Temporal linear relaxation in IBM ILOG CP Optimizer”**. Journal of Scheduling, Nov. 2014.

O. Lhomme. **“Practical Reformulations With Table Constraints”**. ECAI 2012.

P. Vilím. **“Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources”**. CP-AI-OR 2011.

P. Laborie, J. Rogerie, P. Shaw and P. Vilim. **"Reasoning with Conditional Time-intervals - Part II: an Algebraical Model for Resources"**. FLAIRS 2009.

P. Laborie. "**IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems**". CP-AI-OR 2009.

P. Vilim. "**Max Energy Filtering Algorithm for Discrete Cumulative Resources**". CP-AI-OR 2009.

P. Vilim. "**Edge finding filtering algorithm for discrete cumulative resources in *O(knlog(n))***". CP 2009.

P. Laborie and J. Rogerie. "**Reasoning with Conditional Time-intervals**". FLAIRS 2008.

P. Laborie and D. Godard. "**Self-Adapting Large Neighborhood Search: Application to single-mode scheduling problems**". MISTA 2007.

P. Vilim. "**Global Constraints in Scheduling**". PhD thesis. Charles University, Prague. 2007.

P. Laborie, D. Godard and W. Nuijten. **“Randomized Large Neighborhood Search for Cumulative Scheduling”**. ICAPS 2005.

P. Refalo: ”**Impact-Based Search Strategies for Constraint Programming**“. CP 2004.

P. Shaw: "**A Constraint for Bin Packing**".CP 2004.

Agenda

- CP Optimizer overview
- Recent advances
  - Modeling
  - Tools
  - Automatic search
- Q&A

- Lexicographical objectives (12.3)
- OverlapLength expressions (12.2)
- SameSequence constraints (12.6)

- Starting points (old but so useful)
- Conflict refiner (12.5)
- I/O Format (12.6.1)
- Strong annotations (12.6)

- Presolve (12.5)
- Failure-Directed Search (12.6)