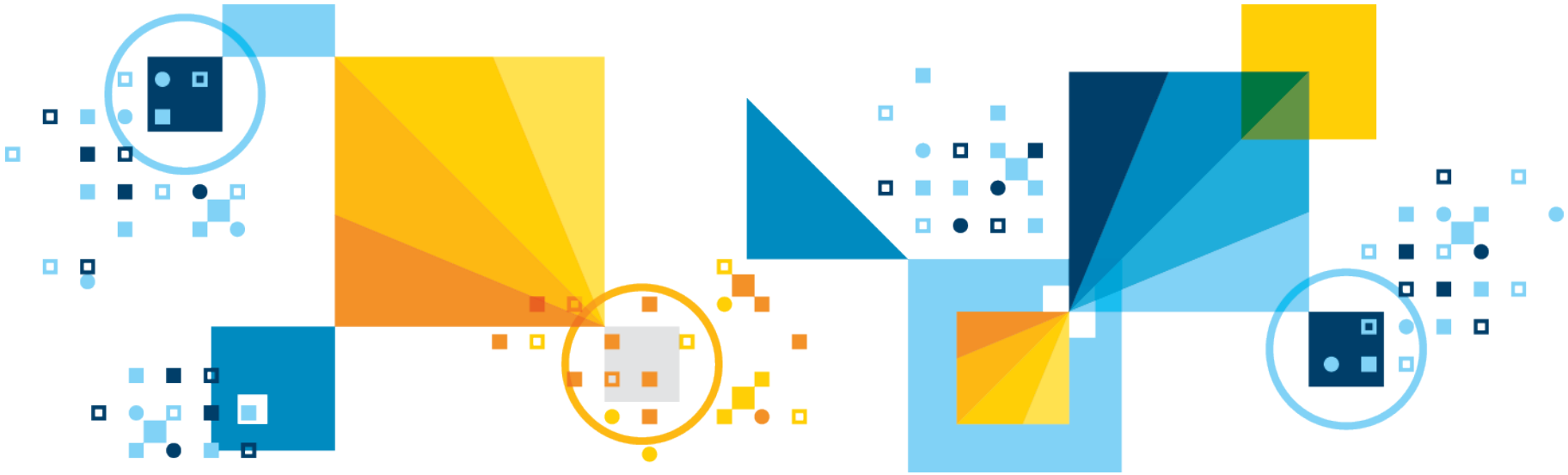
Accelerating the Development of Efficient CP Optimizer Models

Philippe Laborie
IBM Analytics, Decision Optimization



July 14, 2015

Outline

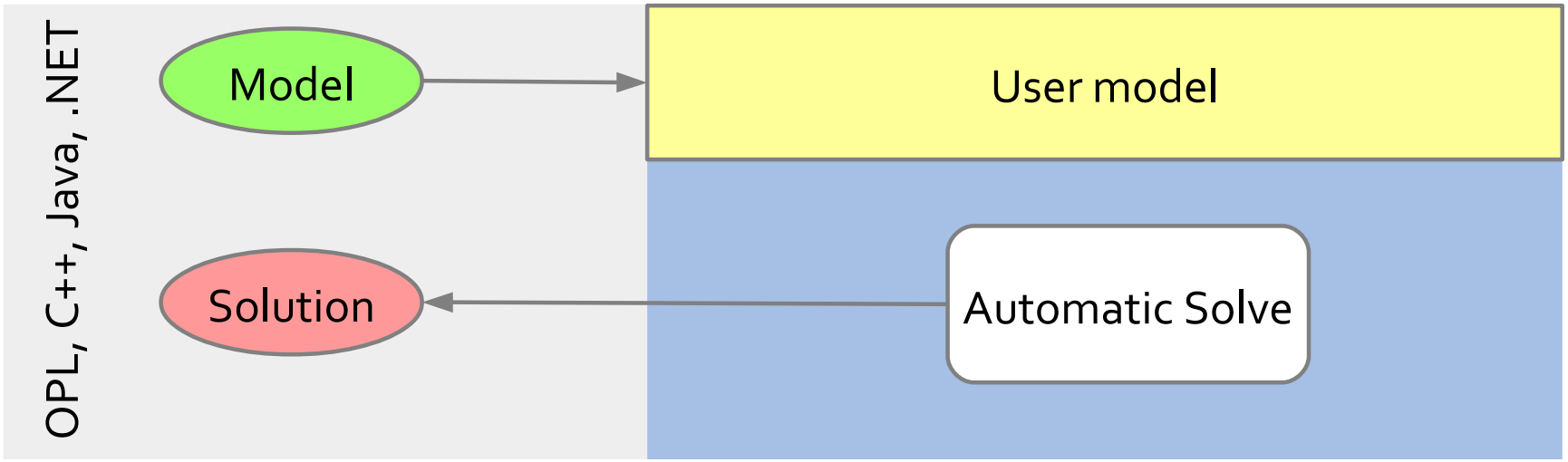
- Overview of CP Optimizer
  - Model&Run paradigm
  - Model
  - Automatic search
- Tools
  - I/O format
  - Model warnings
  - Model presolve
  - Search log
  - Warm start
  - Conflict refiner
  - Failure explainer

Overview of CP Optimizer

- 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)
- Set of **tools** to support the delevopment of efficient models

Overview of CP Optimizer

- Model & Run paradigm



Model : 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 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)

Model : interval variables

- 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 dimension of scheduling problems

Model : interval variables

- In scheduling models, **interval variables** usually represent an interval of time during which some property hold (e.g. an activity executes) and whose end-points (start/end) are decision variables of the problem.
- Examples:
  - A sub-project in a project, a task in a sub-project (Work Breakdown Structure)
  - 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 resource
- Idea of the model (and search) is to avoid the enumeration of start/end values (continuous time)

Model : interval variables

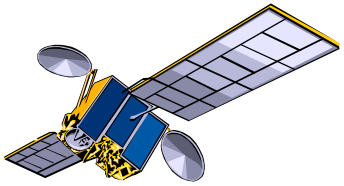
- An interval variable can be **optional** meaning that it is a decision to have it present or absent in a solution.
- Examples:
  - Unperformed tasks and optional sub-projects
  - Alternative resources, modes or recipes for processing an order, each mode specifying a particular combination of operational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

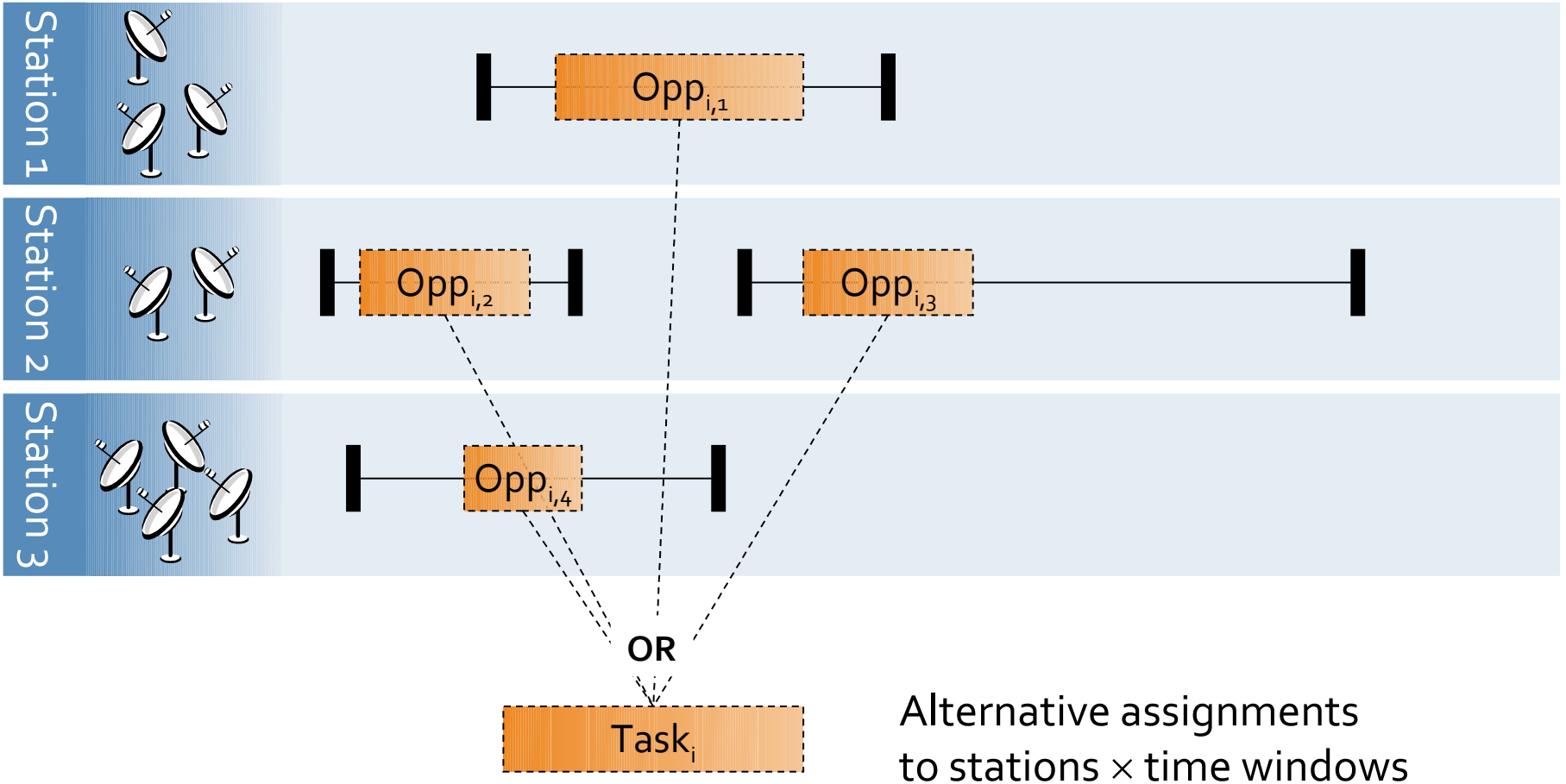
Model: example

- Satellite Control Network scheduling problem [1]
- n communication tasks 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
- Objective: maximize the number of scheduled tasks

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

Model: example





Alternative assignments to stations × time windows (opportunities)

Model: example

```
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] optional;
dvar interval opp[o in Opportunities] optional in o.smin..o.emax size o.dur;

maximize sum(t in Tasks) presenceOf(task[t]);
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;
}
```

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

- 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
  - Opportunistic probing
  - Dominance rules
  - LP-assisted heuristics
  - Randomization
  - Evolutionary algorithms
  - No-good learning
  - Impact-based branching

[2] P. Vilim's talk at this ISMP: "Failure-directed Search for Constraint-based Scheduling"

Tools: I/O format

- Objective:
  - Make it easier to understand the content of a model
  - Communicate a model to IBM support team regardless of the API used to build it (OPL, C++, Java, .NET)
- Structure of a .cpo file
  - Human readable
  - Flat (no cycle, forall statements)
  - No user defined data types
  - Internal information such as CPO version or platform used
  - Includes search parameter values
- Facilities
  - Export model before/instead of solve
  - Export model during solve (with current domains)
  - Import model instead of normal modeling

Tools: I/O format

```
// Interval-related variables:

"task(1)"  = intervalVar(optional);
"task(1A)" = intervalVar(optional);
…
"opp({1,1,62})"  = intervalVar(optional, start=62..intervalmax, end=0..99, size=25);
"opp({1A,1,32})" = intervalVar(optional, start=32..intervalmax, end=0..69, size=33);
…

// Objective:

maximize(sum([presenceOf("task(1)"), presenceOf("task(1A)"),  …]));
…
// Constraints:

alternative("task(1)", ["opp({1,1,62})"], 1);
…
pulse("opp({3,1,58})", 1) + pulse("opp({1,1,62})", 1) +  … <= 4;
…

parameters {
  LogVerbosity = Quiet;
}
```

Tools: model warnings

- Like a compiler, CP Optimizer can analyze the model and print some warnings
  - When there is something suspicious in the model
  - Regardless how the model was created (C++, OPL, …)
  - Including guilty part of the model in the cpo file format
  - Including source code line numbers (if known)
  - 3 levels of warnings, more than 50 types of warnings

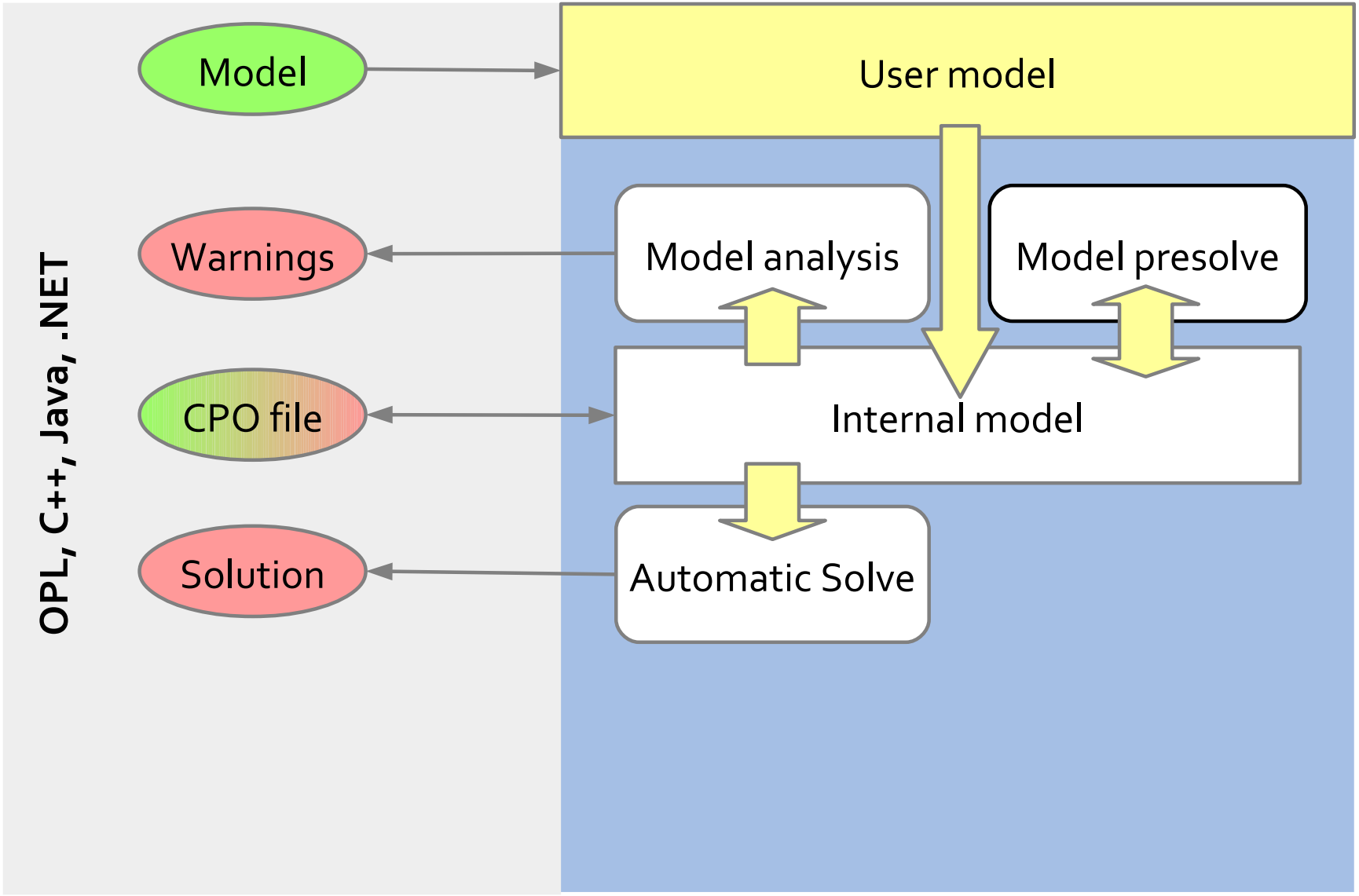
```
satellite.cpo:2995:1: Warning: Constraint 'alternative': there is only one alternative
interval variable.
                    alternative("task(1)", ["opp({1,1,62})"], 1)

satellite.cpo:2996:1: Warning: Constraint 'alternative': there is only one alternative
interval variable.
                    alternative("task(1A)", ["opp({1A,1,32})"], 1)
```

Tools: model presolve

- Objective: **automatically** reformulate the model in order to speed-up its resolution
- Works on an internal representation of the model
- Different types of presolve:
  - Aggregation of basic constraints into global constraints
  - Constraint strengthening
  - Simplifications and factorizations

Tools: model presolve



Tools: model presolve

- Examples of presolve rules
  - Common sub-expression elimination
  - Aggregation of `x!=y` cliques as **`allDifferent`**`([x,y,...])`
  - Precedence strengthening
    - If a and b cannot overlap and **`startsBeforeStart`**`(a,b)`
    - Then **`endsBeforeStart`**`(a,b)`
  - Precedence recognition
    - If **`endOf`**`(a,-∞) ≤` **`startOf`**`(b,+∞)`
    - Then **`endsBeforeStart`**`(a,b)`
    - Precedences are aggregated into a "time net" (STN) for faster and stronger propagation
  - 2-SAT clauses recognition
    - **`presenceOf`**`(a) ≤` **`presenceOf`**`(b)`
    - Such clauses are aggregated into a "logical net" for stronger propagation

Tools: search log

- Objective: understand what happens during the automatic search

```
! ----------------------------------------------------------------------
! Maximization problem - 2980 variables, 853 constraints
! Workers                 = 2
! TimeLimit               = 30
! Initial process time : 0.01s (0.00s extraction + 0.01s propagation)
!  . Log search space  : 4627.3 (before), 4627.3 (after)
!  . Memory usage      : 16.9 MB (before), 19.7 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*           746     3945 0.79s         1            -
            746     4000       2924    1       on task("8")
            746     4000       2908    2       on opp({"186",2,66})
 …
! Time = 1.37s, Explored branches = 35832, Memory usage = 55.5 MB
!          Best Branches  Non-fixed    W       Branch decision
            818    12000       2920    1       on task("184")
 …
! ----------------------------------------------------------------------
! Search terminated by limit, 6 solutions found.
! Best objective         : 826
! Number of branches     : 709092
! Number of fails        : 179648
! Total memory usage     : 54.5 MB (52.9 MB CP Optimizer + 1.6 MB Concert)
! Time spent in solve    : 30.03s (30.01s engine + 0.01s extraction)
! Search speed (br. / s) : 23625.4
! ----------------------------------------------------------------------
```

Problem characteristics

Tools: search log

- Objective: understand what happens during the automatic search

```
! ----------------------------------------------------------------------
! Maximization problem - 2980 variables, 853 constraints
! Workers                 = 2
! TimeLimit               = 30
! Initial process time : 0.01s (0.00s extraction + 0.01s propagation)
!  . Log search space  : 4627.3 (before), 4627.3 (after)
!  . Memory usage      : 16.9 MB (before), 19.7 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*           746     3945 0.79s         1            -
            746     4000       2924    1       on task("8")
            746     4000       2908    2       on opp({"186",2,66})
 …
! Time = 1.37s, Explored branches = 35832, Memory usage = 55.5 MB
!          Best Branches  Non-fixed    W       Branch decision
            818    12000       2920    1       on task("184")
 …
! ----------------------------------------------------------------------
! Search terminated by limit, 6 solutions found.
! Best objective         : 826
! Number of branches     : 709092
! Number of fails        : 179648
! Total memory usage     : 54.5 MB (52.9 MB CP Optimizer + 1.6 MB Concert)
! Time spent in solve    : 30.03s (30.01s engine + 0.01s extraction)
! Search speed (br. / s) : 23625.4
! ----------------------------------------------------------------------
```

Modified parameter values

Tools: search log

- Objective: understand what happens during the automatic search

```
! ----------------------------------------------------------------------
! Maximization problem - 2980 variables, 853 constraints
! Workers                 = 2
! TimeLimit               = 30
! Initial process time : 0.01s (0.00s extraction + 0.01s propagation)
!  . Log search space  : 4627.3 (before), 4627.3 (after)
!  . Memory usage      : 16.9 MB (before), 19.7 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*           746     3945 0.79s         1            -
            746     4000       2924    1       on task("8")
            746     4000       2908    2       on opp({"186",2,66})
 …
! Time = 1.37s, Explored branches = 35832, Memory usage = 55.5 MB
!          Best Branches  Non-fixed    W       Branch decision
            818    12000       2920    1       on task("184")
 …
! ----------------------------------------------------------------------
! Search terminated by limit, 6 solutions found.
! Best objective         : 826
! Number of branches     : 709092
! Number of fails        : 179648
! Total memory usage     : 54.5 MB (52.9 MB CP Optimizer + 1.6 MB Concert)
! Time spent in solve    : 30.03s (30.01s engine + 0.01s extraction)
! Search speed (br. / s) : 23625.4
! ----------------------------------------------------------------------
```

Root node information

Tools: search log

- Objective: understand what happens during the automatic search

```
! ----------------------------------------------------------------------
! Maximization problem - 2980 variables, 853 constraints
! Workers               = 2
! TimeLimit             = 30
! Initial process time : 0.01s (0.00s extraction + 0.01s propagation)
!  . Log search space  : 4627.3 (before), 4627.3 (after)
!  . Memory usage      : 16.9 MB (before), 19.7 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------
!          Best Branches  Non-fixed    W       Branch decision
*           746     3945 0.79s         1            -
            746     4000       2924    1       on task("8")
            746     4000       2908    2       on opp({"186",2,66})
 …
! Time = 1.37s, Explored branches = 35832, Memory usage = 55.5 MB
!          Best Branches  Non-fixed    W       Branch decision
            818    12000       2920    1       on task("184")
 …
! ----------------------------------------------------------------------
! Search terminated by limit, 6 solutions found.
! Best objective         : 826
! Number of branches     : 709092
! Number of fails        : 179648
! Total memory usage     : 54.5 MB (52.9 MB CP Optimizer + 1.6 MB Concert)
! Time spent in solve    : 30.03s (30.01s engine + 0.01s extraction)
! Search speed (br. / s) : 23625.4
! ----------------------------------------------------------------------
```

New incumbent solutions (time, worker)

Tools: search log

- Objective: understand what happens during the automatic search

```
! ----------------------------------------------------------------------
! Maximization problem - 2980 variables, 853 constraints
! Workers                 = 2
! TimeLimit               = 30
! Initial process time : 0.01s (0.00s extraction + 0.01s propagation)
!  . Log search space  : 4627.3 (before), 4627.3 (after)
!  . Memory usage      : 16.9 MB (before), 19.7 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------
!           Best Branches  Non-fixed    W       Branch decision
*            746     3945 0.79s         1            -
             746     4000       2924    1       on task("8")
             746     4000       2908    2       on opp({"186",2,66})
 …
! Time = 1.37s, Explored branches = 35832, Memory usage = 55.5 MB
!           Best Branches  Non-fixed    W       Branch decision
             818    12000       2920    1       on task("184")
 …
! ----------------------------------------------------------------------
! Search terminated by limit, 6 solutions found.
! Best objective         : 826
! Number of branches     : 709092
! Number of fails        : 179648
! Total memory usage     : 54.5 MB (52.9 MB CP Optimizer + 1.6 MB Concert)
! Time spent in solve    : 30.03s (30.01s engine + 0.01s extraction)
! Search speed (br. / s) : 23625.4
! ----------------------------------------------------------------------
```

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

Tools: search log

- Objective: understand what happens during the automatic search

```
! ----------------------------------------------------------------------
! Maximization problem - 2980 variables, 853 constraints
! Workers                 = 2
! TimeLimit               = 30
! Initial process time : 0.01s (0.00s extraction + 0.01s propagation)
!  . Log search space  : 4627.3 (before), 4627.3 (after)
!  . Memory usage      : 16.9 MB (before), 19.7 MB (after)
! Using parallel search with 2 workers.
! ----------------------------------------------------------------------
!           Best Branches  Non-fixed    W       Branch decision
*            746     3945 0.79s         1            -
             746     4000       2924    1       on task("8")
             746     4000       2908    2       on opp({"186",2,66})
 …
! Time = 1.37s, Explored branches = 35832, Memory usage = 55.5 MB
!           Best Branches  Non-fixed    W       Branch decision
             818    12000       2920    1       on task("184")
 …
! ----------------------------------------------------------------------
! Search terminated by limit, 6 solutions found.
! Best objective         : 826
! Number of branches     : 709092
! Number of fails        : 179648
! Total memory usage     : 54.5 MB (52.9 MB CP Optimizer + 1.6 MB Concert)
! Time spent in solve    : 30.03s (30.01s engine + 0.01s extraction)
! Search speed (br. / s) : 23625.4
! ----------------------------------------------------------------------
```

Final information with solution status and search statistics

Tools: warm start

- 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: warm start

- Satellite Control Network scheduling problem [1]
- n communication tasks 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
- Tasks have priorities: first maximize the number of scheduled high priority tasks, then the number of scheduled low priority tasks

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

Tools: warm start

```
50  // --------------------------------------------------------------------
51  // STEP 1: MAXIMIZE NUMBER OF SCHEDULED HIGH-PRIORITY TASKS
52  var opl1 = new IloOplModel(def, cp);
53  // Maximize number of high priority tasks:
54  data.BestHighPriorities = -1;
55  opl1.addDataSource(data);
56  opl1.generate();
57  cp.solve();
58
59  // --------------------------------------------------------------------
60  // STEP 2: MAXIMIZE NUMBER OF SCHEDULED LOW-PRIORITY TASKS
61  var cp2 = new IloCP();
62  var opl2 = new IloOplModel(def, cp2);
63  // Maximize number of low priority tasks:
64  data.BestHighPriorities = opl1.nbHighPriorities;
65  opl2.addDataSource(data);
66  opl2.generate();
67
68  // SETTING STARTING POINT
69  var sp = new IloOplCPSolution();
70  sp.setPresence(opl2.opp, opl1.opp);
71  sp.setStart(opl2.opp, opl1.opp);
72  cp2.setStartingPoint(sp);
73  cp2.solve();
```

Tools: conflict refiner

- Objective: identify a reason for an inconsistency by providing a **minimal infeasible subset** of constraints for an infeasible model
- 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

Tools: conflict refiner

```
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] optional 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;
}
```

Tools: conflict refiner

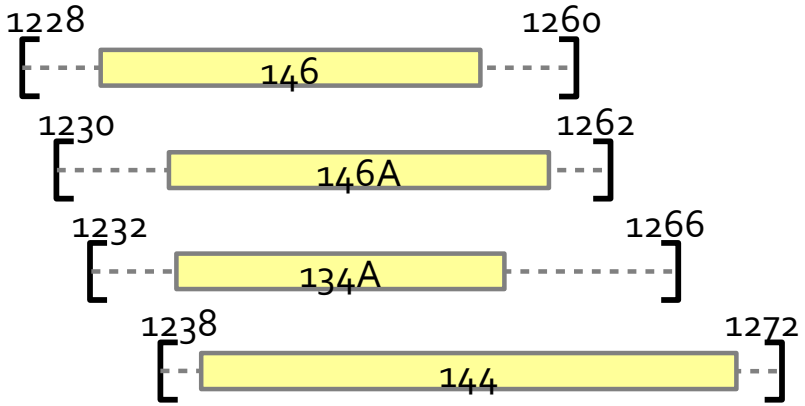
```
!--------------------------------------------------------------------------
! Satisfiability problem - 2,980 variables, 851 constraints
! Problem found infeasible at the root node
! ------------------------------------------------------------------------
  ...
! ------------------------------------------------------------------------
! 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
! ------------------------------------------------------------------------
```

Tools: conflict refiner

- 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>



Tools: failure explainer

- Objective: explains why the engine backtracks at a given search node
- Uses the conflict refiner to find a minimal conflict in the model+decisions at a backtracking node
- Currently only available in DepthFirst search mode and only for integer variables

Tools: failure explainer

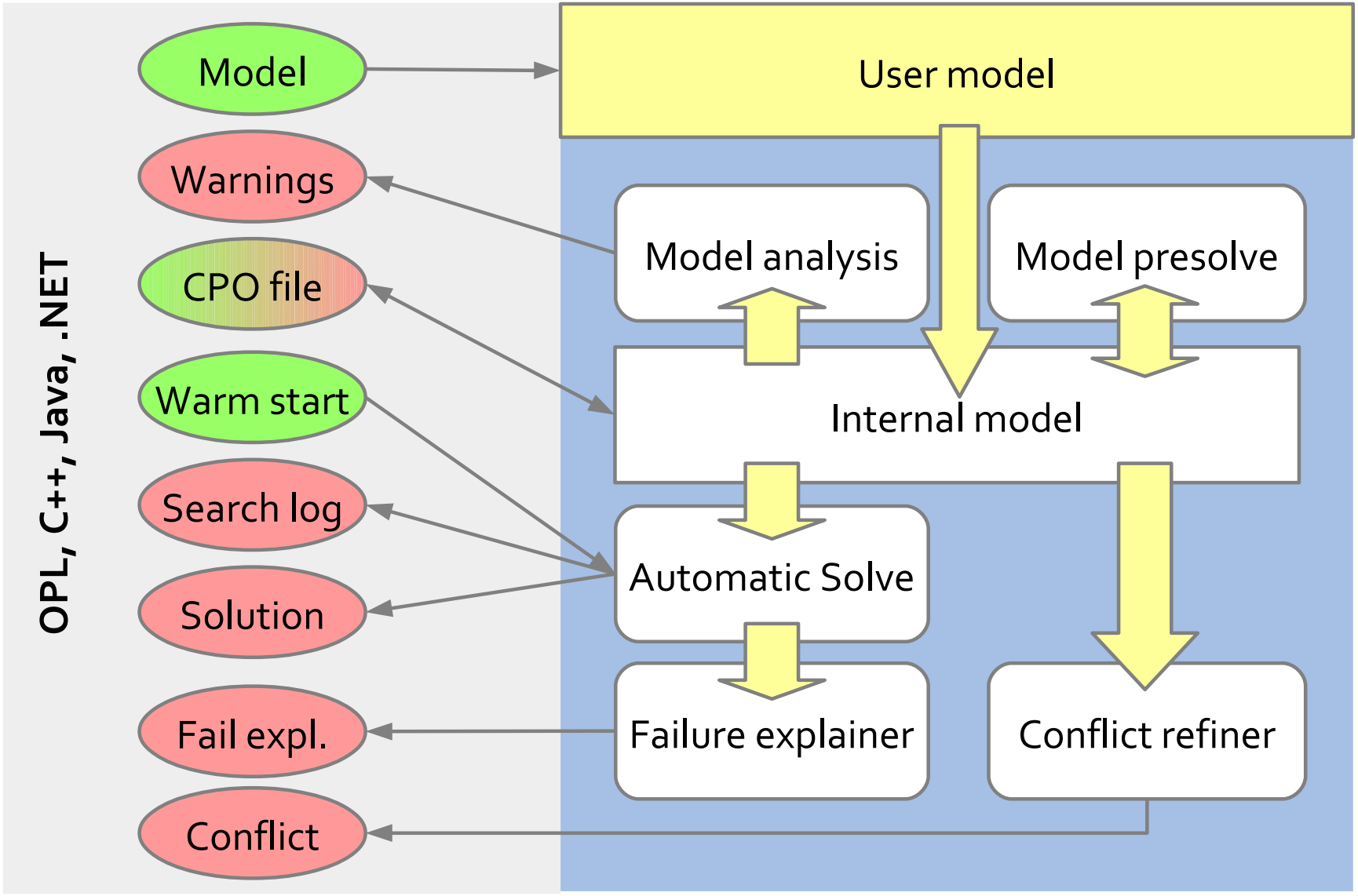
- Example

```
// Build model:
…
// Create CP object:
IloCP cp(model);
// Use only one thread:
cp.setParameter(IloCP::Workers, 1);
// Simple tree search:
cp.setParameter(IloCP::SearchType, IloCP::DepthFirst);
// Show failure numbers:
cp.setParameter(IloCP::LogSearchTags, IloCP::On);
// Explain particular failures:
cp.explainFailure(IloIntArray(env, 4, 3, 10, 11, 12));
// Solve and explain:
cp.solve();
```

Tools: failure explainer

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(store1, [location1, location2, location3, location4, location5]) == 1;
element(store2, [location1, location2, location3, location4, location5]) == 1;
element(store3, [location1, location2, location3, location4, location5]) == 1;
element(store4, [location1, location2, location3, location4, location5]) == 1;
element(store5, [location1, location2, location3, location4, location5]) == 1;
element(store6, [location1, location2, location3, location4, location5]) == 1;
element(store7, [location1, location2, location3, location4, location5]) == 1;
element(store8, [location1, location2, location3, location4, location5]) == 1;
count([store1, store2, store3, store4, store5, store6, store7, store8], 0) <= 3;
count([store1, store2, store3, store4, store5, store6, store7, store8], 3) <= 4;
// Branch constraints
location2 == 0;
location3 == 0;
location5 == 0;
```

The full picture



Questions ?