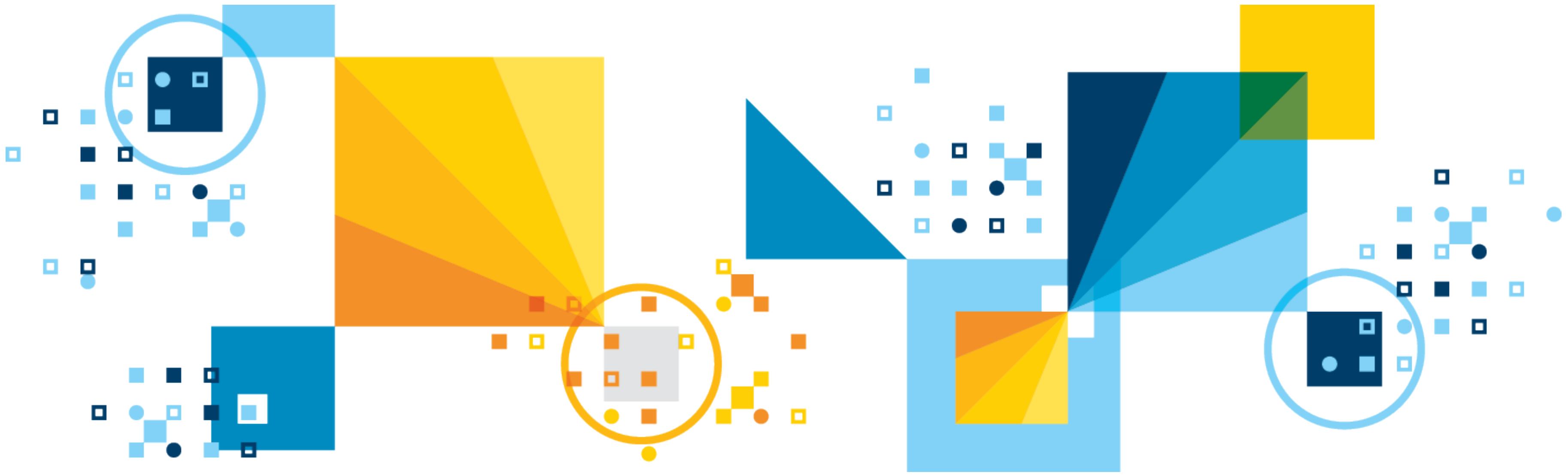
Insights into CP Optimizer models: Using the tools



Nov. 28, 2016

Outline

- Overview of CP Optimizer
- Tools providing some insight into models and resolution
  - I/O format
  - Model warnings
  - Search log
  - Warm start
  - Conflict refiner
  - Failure explainer

Overview of CP Optimizer

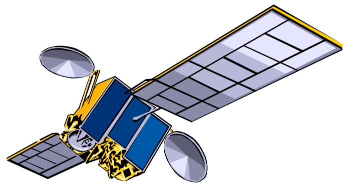
- A component of **IBM ILOG CPLEX Optimization Studio**
- A **Constraint Programming** engine for combinatorial problems (including detailed scheduling problems)
- Implements a **Model & Run** paradigm (like CPLEX)
  - Model: **Concise** yet **expressive** modeling language
  - Run: **Powerful automatic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

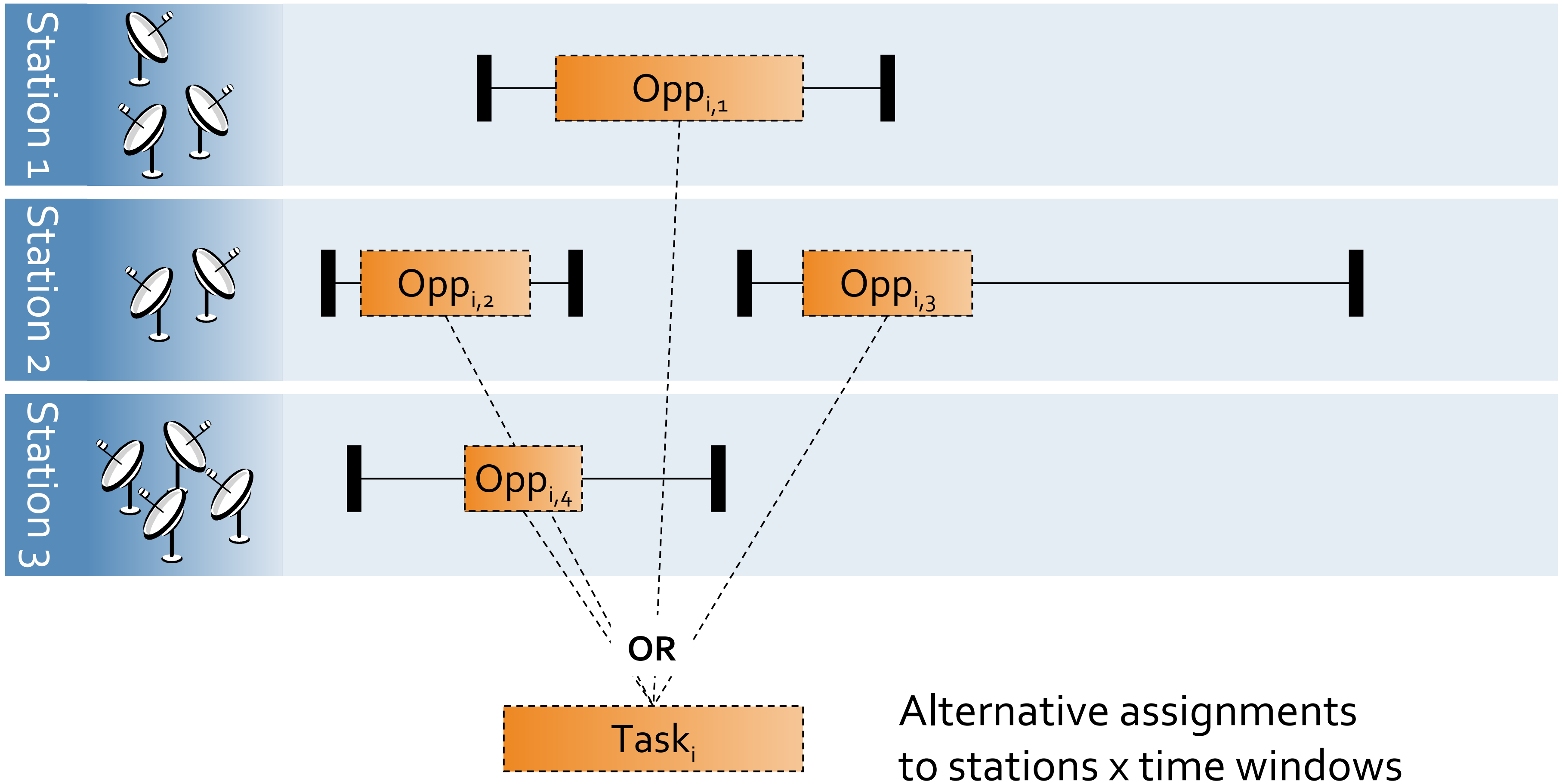
Model: example

- 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 x time windows (opportunities)

Model: example (OPL)

```
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

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)
  - Send model to a remote CPO engine (e.g. Cloud)
- Structure of a .cpo file
  - Human readable
  - Flat (no cycle, no forall statements)
  - No user defined data types
  - Internal information such as CPO version or platform used
  - Includes search phases, search parameter values and starting point
- Engine Functionality
  - 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++, Java, Python, OPL, …)
  - Including guilty part of the model in the cpo file format (optional)
  - Including source code line numbers (if known)
  - 3 levels of warnings, more than 70 types of warnings

```
cppfile.cpp:24: Warning: Unused interval variable 'x'.
      x = intervalVar(start=1..50, size=5..10)

javafile.java:20: Warning: Interval variable 'itv' has empty domain.
      itv = intervalVar(start=0..10, length=5, end=100..110)

pythonfile.py:7: Warning: Constraint is always true.
      x+y >= 5

pythonfile.py:8: Warning: Constraint is always false, the model is infeasible.
      x+y < 5

satellite.cpo:2995:29: Warning: Constraint 'alternative':
                        there is only one alternative interval variable.
      alternative("task(1)", ["opp({1,1,62})"], 1)
```

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

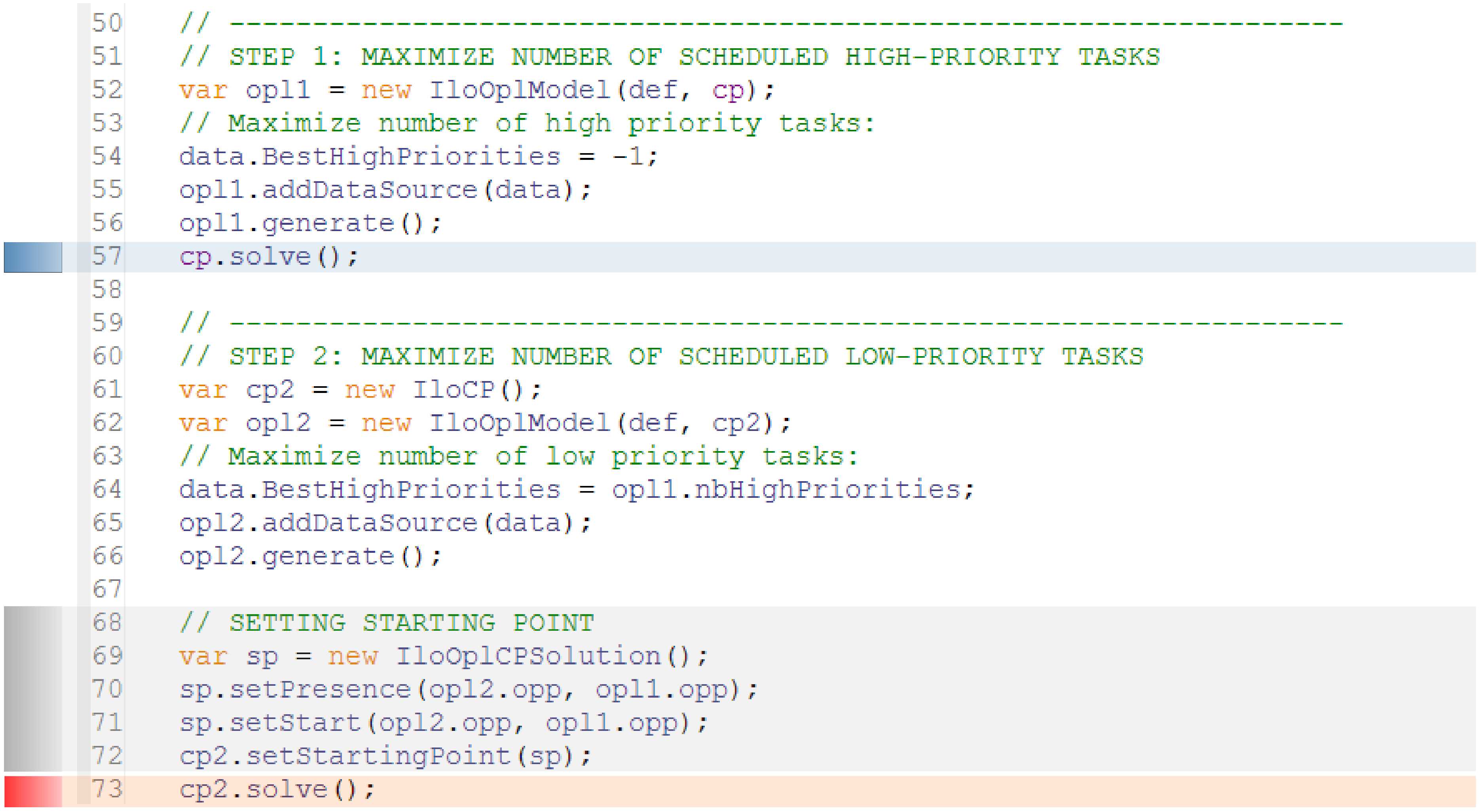
- Objective: Start search from a known (possibly incomplete) solution given by the user (warm start) in order to further improve it or to help to guide the engine towards a first feasible solution
- API: `IloCP::setStartingPoint(IloSolution warmstart)`
- Use cases:
  - Restart an **interrupted search** 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]
- 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, number of tasks from to 400 to 1300
- Tasks are not mandatory but have priorities
- Hierarchical objectives:
  - 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 (OPL sample)

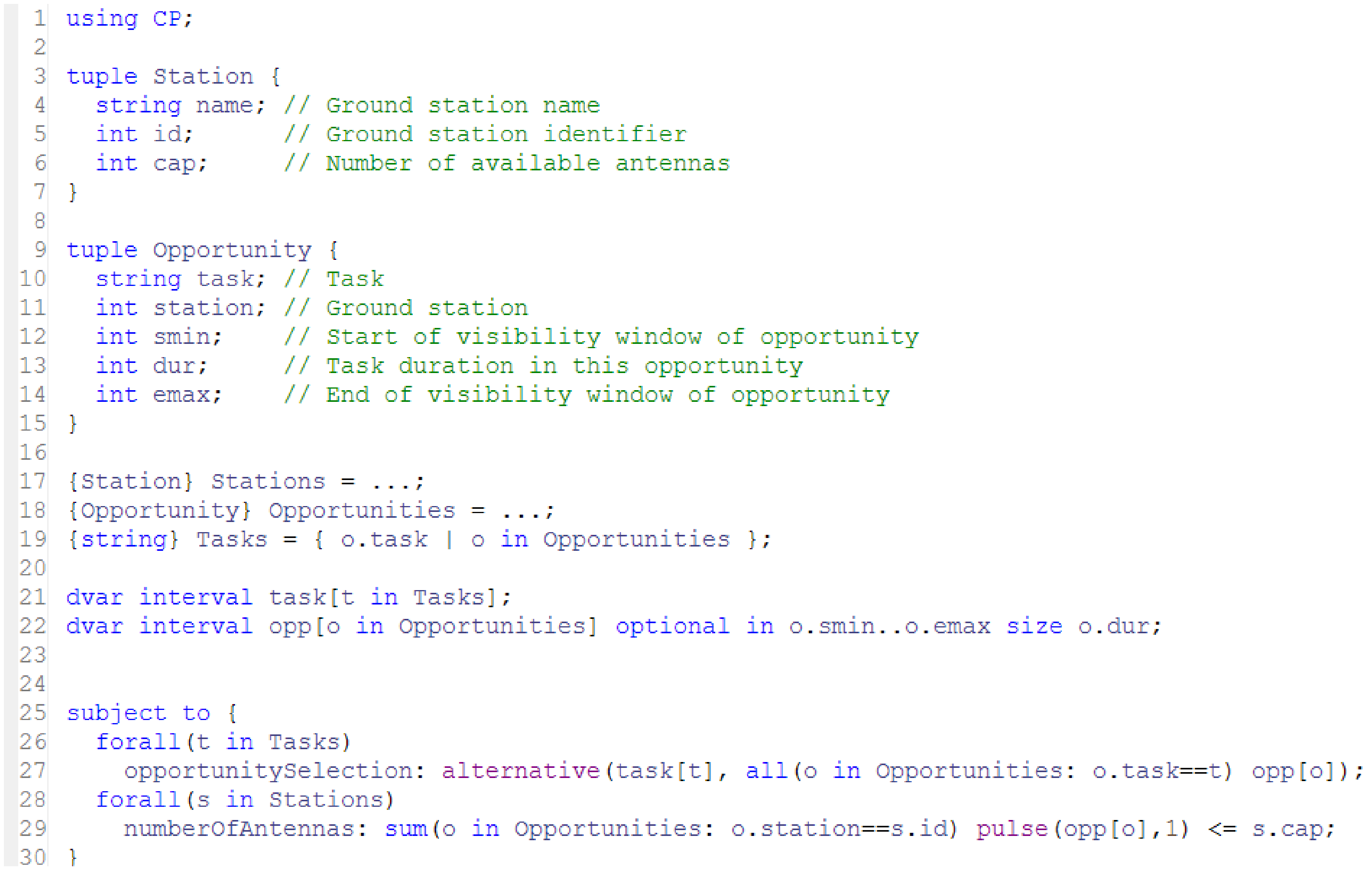


```
// ------------------------------------------------------------------
// STEP 1: MAXIMIZE NUMBER OF SCHEDULED HIGH-PRIORITY TASKS
var opl1 = new IloOplModel(def, cp);
// Maximize number of high priority tasks:
data.BestHighPriorities = -1;
opl1.addDataSource(data);
opl1.generate();
cp.solve();

// ------------------------------------------------------------------
// STEP 2: MAXIMIZE NUMBER OF SCHEDULED LOW-PRIORITY TASKS
var cp2 = new IloCP();
var opl2 = new IloOplModel(def, cp2);
// Maximize number of low priority tasks:
data.BestHighPriorities = opl1.nbHighPriorities;
opl2.addDataSource(data);
opl2.generate();

// SETTING STARTING POINT
var sp = new IloOplCPSolution();
sp.setPresence(opl2.opp, opl1.opp);
sp.setStart(opl2.opp, opl1.opp);
cp2.setStartingPoint(sp);
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



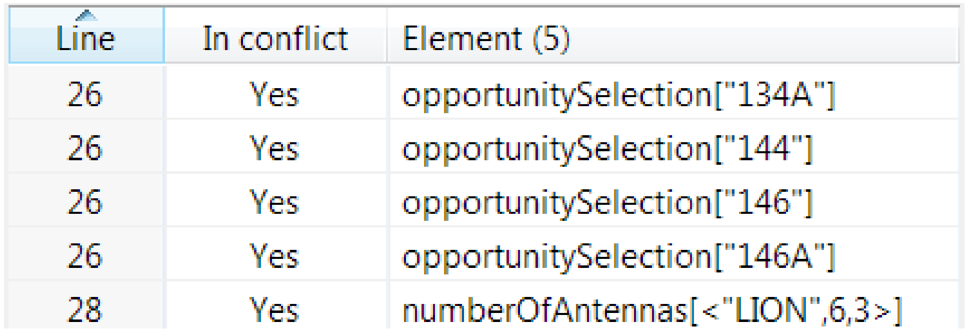
```
using CP;

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
! ----------------------------------------------------------------------------
```

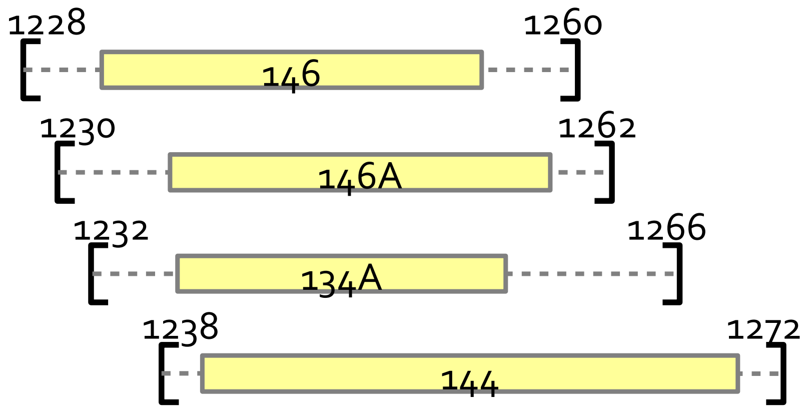
Tools: conflict refiner (OPL display)

- 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: conflict refiner

- Advanced features:
  - Control which conflict is selected by assigning some priorities to each constraint
  - Run conflict refiner on groups of constraints (one group = one business constraint) instead of individual ones for more informative conflicts

- Control parameters :
  - ConflictRefinerOnVariables : Off/On
  - ConflictRefinerIterationLimit
  - ConflictRefinerBranchLimit
  - ConflictRefinerFailLimit
  - ConflictRefinerTimeLimit

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 – first step

- Solve the model in depth first search in a mode that display failure index and decisions:

```
// 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);
// Solve and display the failure:
cp.solve();
```

Tools: failure explainer – second step

- Solve the model in depth first search, specifying the index of the failures to explain :

```
// 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);
 // Explain particular failures:
 cp.explainFailure(IloIntArray(env, 4, 3, 10, 11, 12));
 // Solve and explain:
 cp.solve();
```

Tools: failure explainer example: assigning location to store

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(loc_0, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_1, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_2, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_3, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_4, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_6, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_7, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_8, [open_0, open_1, open_2, open_3, open_4]) == 1;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 0) <= 3;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 3) <= 4;
// Branch constraints
open_1 == 0;
open_2 == 0;
open_4 == 0;
```

Tools: failure explainer example: assigning location to store

loc_x: location of store x, value in <0, 4>

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(loc_0, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_1, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_2, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_3, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_4, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_6, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_7, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_8, [open_0, open_1, open_2, open_3, open_4]) == 1;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 0) <= 3;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 3) <= 4;
// Branch constraints
open_1 == 0;
open_2 == 0;
open_4 == 0;
```

Tools: failure explainer example: assigning location to store

open_y: is location y open?, Boolean domain [0,1]

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(loc_0, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_1, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_2, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_3, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_4, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_6, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_7, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_8, [open_0, open_1, open_2, open_3, open_4]) == 1;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 0) <= 3;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 3) <= 4;
// Branch constraints
open_1 == 0;
open_2 == 0;
open_4 == 0;
```

Tools: failure explainer example: assigning location to store

A store should assigned one location among the open ones

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(loc_0, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_1, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_2, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_3, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_4, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_6, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_7, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_8, [open_0, open_1, open_2, open_3, open_4]) == 1;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 0) <= 3;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 3) <= 4;
// Branch constraints
open_1 == 0;
open_2 == 0;
open_4 == 0;
```

Tools: failure explainer example: assigning location to store

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(loc_0, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_1, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_2, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_3, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_4, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_6, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_7, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_8, [open_0, open_1, open_2, open_3, open_4]) == 1;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 0) <= 3;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 3) <= 4;
// Branch constraints
open_1 == 0;
open_2 == 0;
open_4 == 0;
```

At most 3 stores at location 0
At most 4 stores at location 3

Tools: failure explainer example: assigning location to store

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(loc_0, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_1, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_2, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_3, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_4, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_6, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_7, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_8, [open_0, open_1, open_2, open_3, open_4]) == 1;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 0) <= 3;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 3) <= 4;
// Branch constraints
open_1 == 0;
open_2 == 0;
open_4 == 0;
```

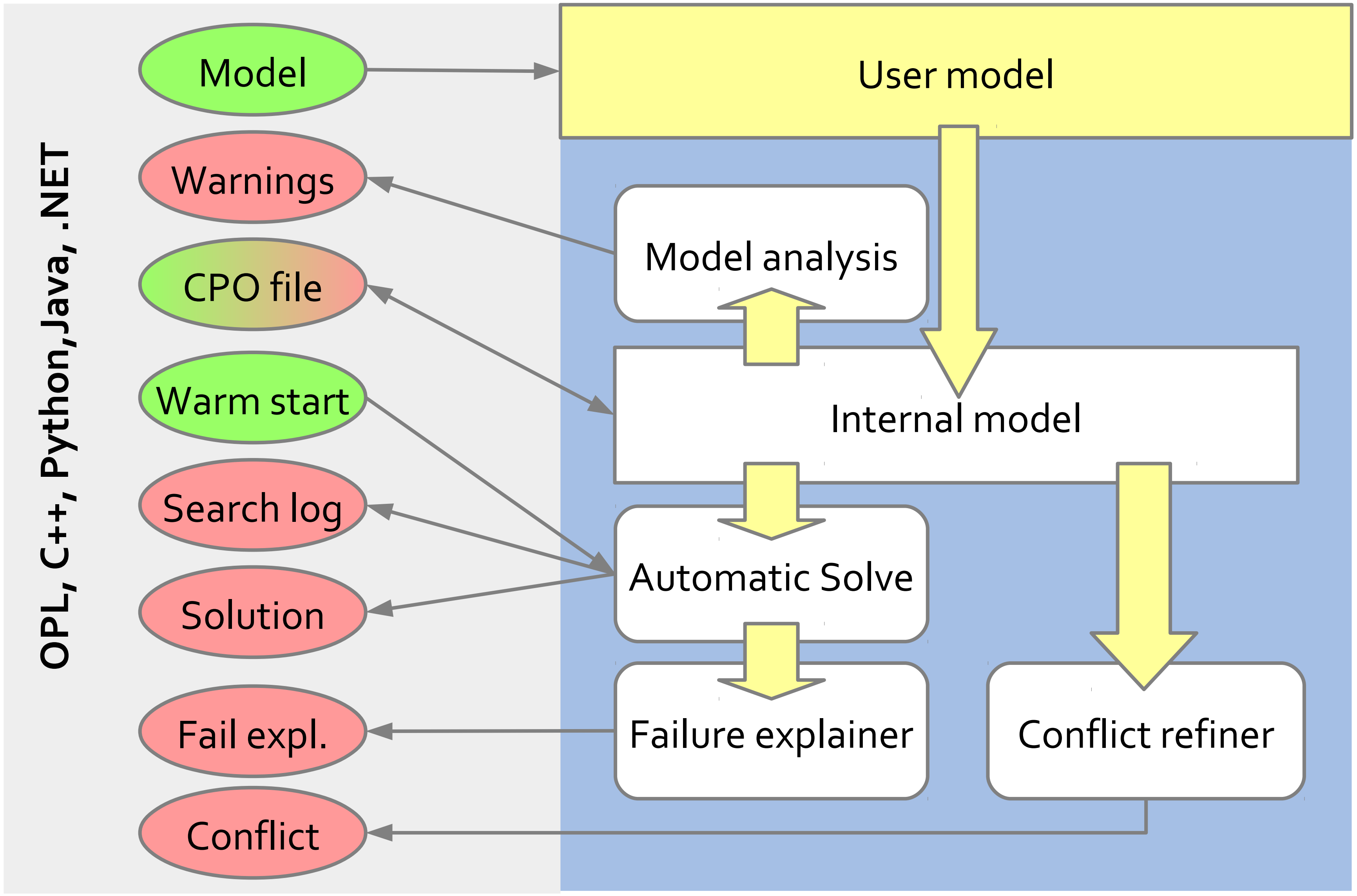
Tools: failure explainer example: assigning location to store

```
 - Failure #1
 - Failure #2
 - Failure #3
-- Possible conflict explaining failure
// Model constraints
element(loc_0, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_1, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_2, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_3, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_4, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_6, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_7, [open_0, open_1, open_2, open_3, open_4]) == 1;
element(loc_8, [open_0, open_1, open_2, open_3, open_4]) == 1;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 0) <= 3;
count([loc_0, loc_1, loc_2, loc_3, loc_4, loc_6, loc_7, loc_8], 3) <= 4;
// Branch constraints
open_1 == 0;
open_2 == 0;
open_4 == 0;
```

At least 9 stores at location 0 or 3

At most 3 stores at location 0
At most 4 stores at location 3

The full picture


OPL, C++, Python,Java, .NET
Model
Warnings
CPO file
Warm start
Search log
Solution
Fail expl.
Conflict
User model
Model analysis
Internal model
Automatic Solve
Failure explainer
Conflict refiner