Résolution de problèmes d'ordonnancement complexes

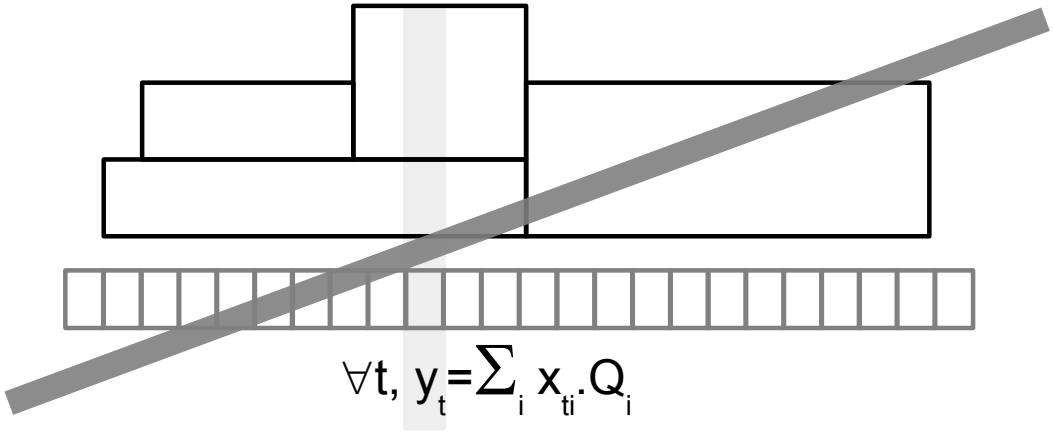
Philippe Laborie, Equipe de développement CP Optimizer

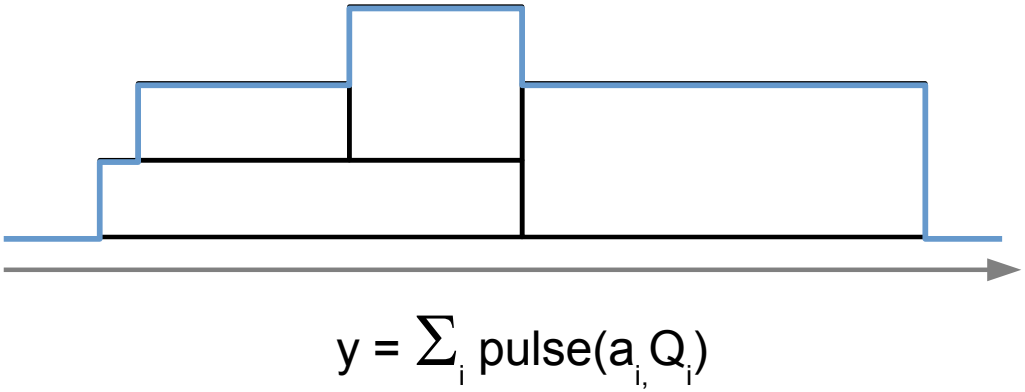
Résolution de problèmes d'ordonnancement complexes

- Overview de CP Optimizer pour l'ordonnancement
  - Modèle
  - Résolution
- Résolution de problèmes d'ordonnancement complexes
  - Quelques éléments de méthodologie
  - Outils d'analyse
    - Search log
    - Conflict refiner
  - Outils pour la résolution
    - Paramètres
    - Warm-start

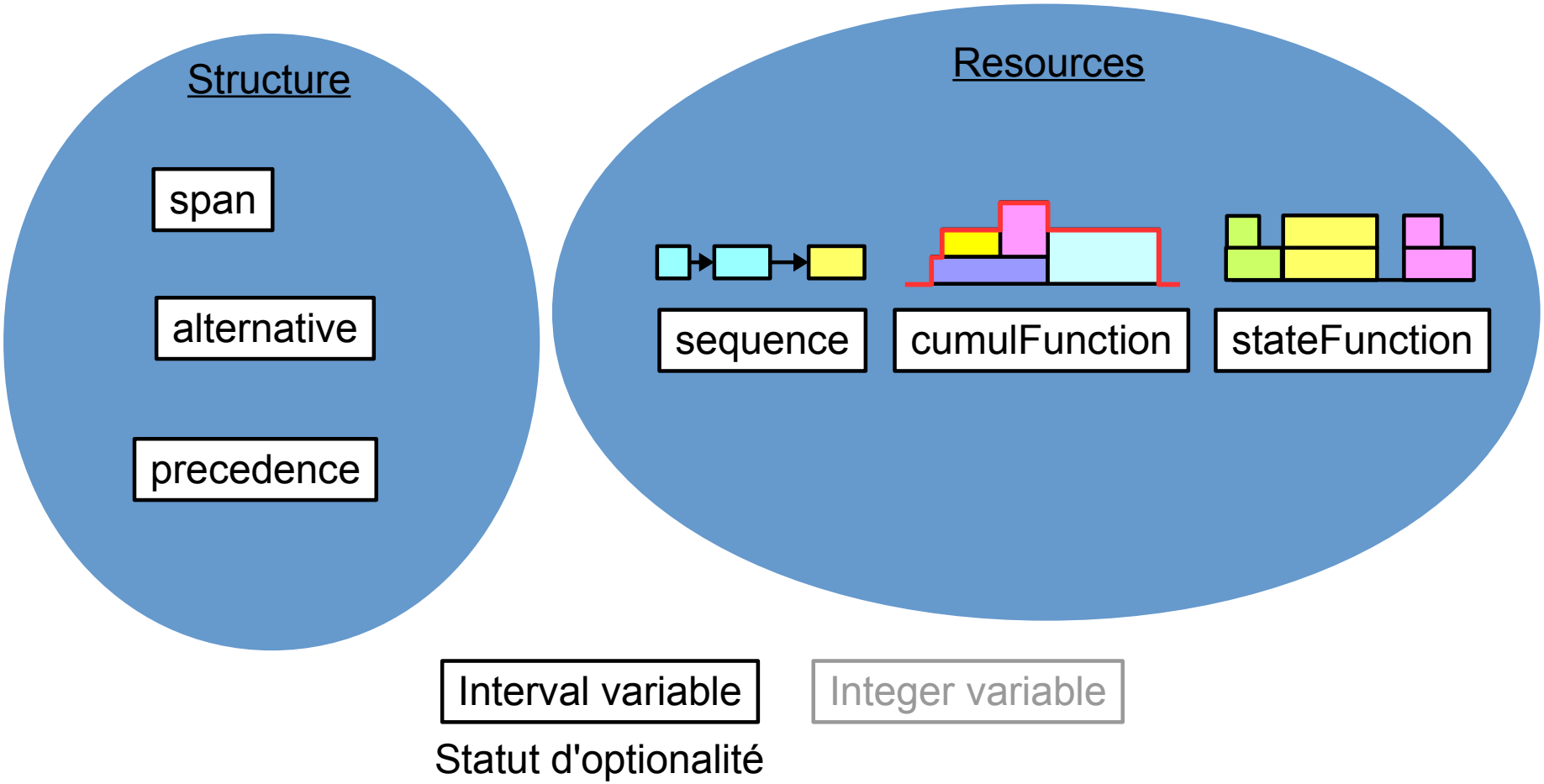
CP Optimizer pour l'ordonnancement / Modèle

- Idée principale: **un modèle qui évite l'énumération du temps**





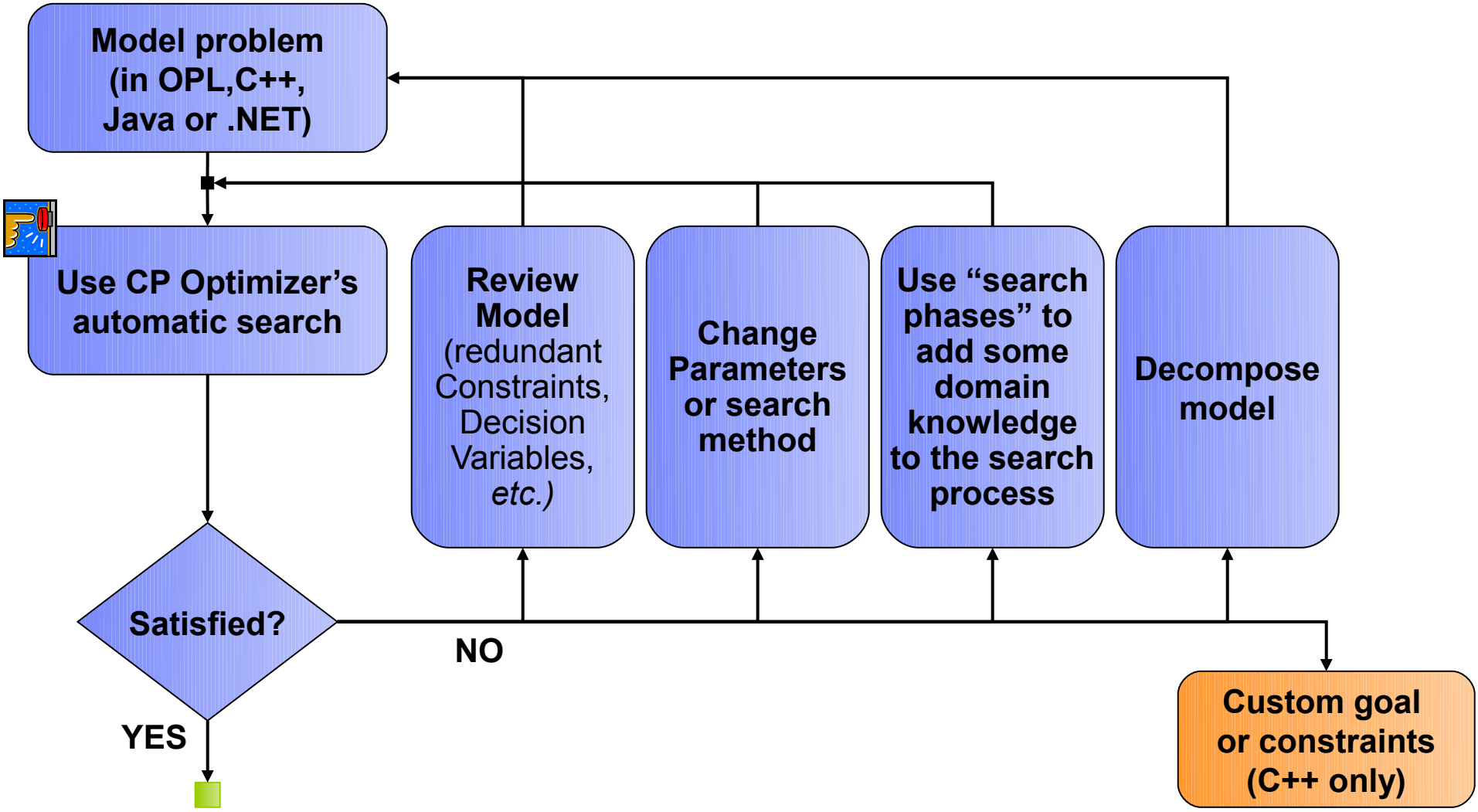
CP Optimizer pour l'ordonnancement / Modèle



CP Optimizer pour l'ordonnancement / Modèle

```
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;
}
```

CP Optimizer pour l'ordonnancement / Résolution



Résolution de problèmes d'ordonnancement complexes

- Quelques éléments de méthodologie
  - Exploiter la richesse du modèle d'ordonnancement
  - Ne pas surexploiter la richesse du modèle d'ordonnancement
  - Privilégier les modèles *faciles*
  - Travailler le plus tôt possible avec des données de taille réelle
  - Ne pas sur-optimiser les modèles

Résolution de problèmes d'ordonnancement complexes

- Exploiter la richesse du modèle d'ordonnancement pour
  - Eviter l'énumération explicite du temps dans le modèle (∀t)
  - Eviter les modèles dont le nombre de variables/contraintes croît plus que linéairement avec une dimension du problème (nombre d'activités, nombre de ressources, …)
    - `forall(a,b … : condition(a,b))`
      `(endOf(a) <= startOf(b)) || (endOf(b) <= startOf(a))`
      `stateFunction …`
  - Eviter les méta-contraintes
    - `presenceOf(a) && presenceOf(b) => endOf(a) <= startOf(b)`
      `endBeforeStart(a,b)`
    - `(endOf(a) <= startOf(b)) || (endOf(b) <= startOf(a))`
      `overlapLength(a,b)==0`
    - `dexpr int tardiness = presenceOf(a)*maxl(0, dd-endOf(a))`
      `dexpr int tardiness = max(0, dd-endOf(a,dd))`
  - Eviter les variables de décisions inutiles

Résolution de problèmes d'ordonnancement complexes

- Exploiter la richesse du modèle d'ordonnancement
- Règle:

  « Les modèles les plus courts sont en général les meilleurs. »

Résolution de problèmes d'ordonnancement complexes

- Ne pas surexploiter la richesse du modèle d'ordonnancement
  - La richesse du langage de modélisation ne doit pas masquer le fait que la plupart des problèmes d'ordonnancement sont **NP-complets**
  - CP Optimizer est efficace pour gérer :
    - Les dates de début/fin des activités (Quand?)
    - Les choix discrets sur leur mode d'exécution: activités optionnelles, ressources alternatives, recettes alternatives (Comment?)
    - Les ressources complexes
  - CP Optimizer est moins efficace pour gérer:
    - Les quantités de ressources (Combien? Pour qui?)
    - La constitution de batchs complexes avec sous-problèmes de packing
      → Pour ces problèmes, penser à une décomposition exploitant la complémentarité entre CPLEX et CP Optimizer

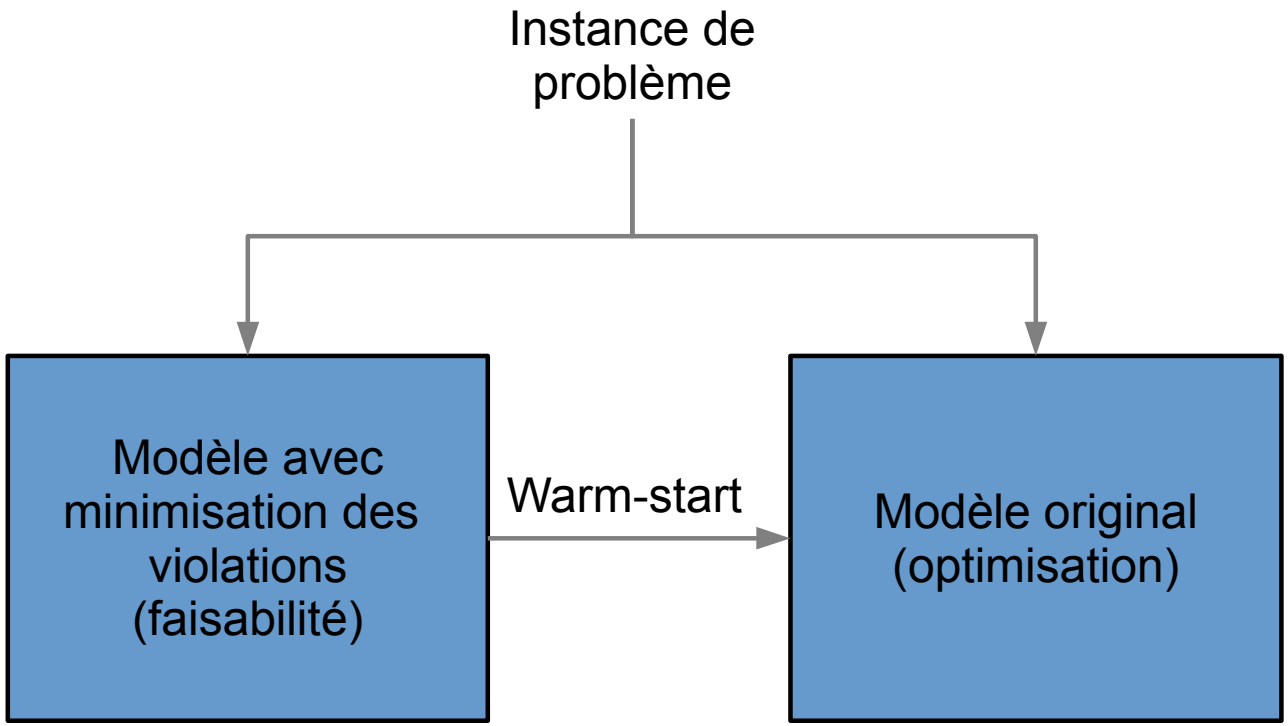
Résolution de problèmes d'ordonnancement complexes

- Ne pas surexploiter la richesse du modèle d'ordonnancement
  - La richesse du langage de modélisation ne doit pas masquer le fait que la plupart des problèmes d'ordonnancement sont **NP-complets**
  - Un modèle n'est pas une description exacte de la réalité. Certains aspects du problème réels doivent être:
    - **Ignorés**
    - **Relâchés**:
      ex: n machines similaires → une ressource de capacité n
    - **Sur-contraints**:
      ex: augmenter la durée d'une activité pour absorber des aléas
    - **Approximés**:
      ex: pour modéliser une fonction objectif implicite
    - **Hiérarchisés**, dans le but de séparer les sources de complexité du problème
    - **Décomposés**, dans le but de résoudre des modèles plus petits

Résolution de problèmes d'ordonnancement complexes

- Privilégier les modèles « faciles »
  - Modèle « facile » = modèles pour lesquels trouver une solution initiale faisable (même de mauvaise qualité) est facile
  - Souvent, les problèmes d'ordonnancement sont « faciles »: un ordonnancement consistant à ne rien faire est faisable (mais mauvais!)
  - … la difficulté vient de la fonction objectif: maximiser les profits (activités, production réalisée, ...), minimiser les coûts (avance/retard, ressources, …)
- CP Optimizer est en général plus efficace pour améliorer une solution existante que pour trouver une solution à un problème de satisfiabilité difficile
- → pour trouver une solution faisable initiale il peut être intéressant de faire passer des contraintes dans la fonction objectif

Résolution de problèmes d'ordonnancement complexes



Violations (en général, facile à modéliser dans CP Optimizer):

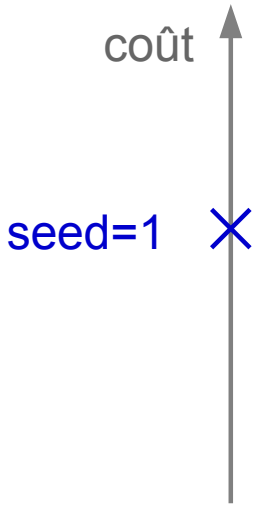
- Activités non-réalisées (optionalité)
- Retards
- Précédences
- Utilisation de ressource

Résolution de problèmes d'ordonnancement complexes

- Travailler le plus tôt possible avec des données de taille réelle
  - Il est très difficile de qualifier *a priori* comment un modèle donné va passer à l'échelle. Pour une qualité de solution donnée, le comportement en fonction d'une dimension *n* du problème peut varier en:
    - *O(n)*: problème facile avec peu de dépendances entre décisions
    - $O(n^2)$: problème facile avec dépendances entre décisions
    - $O(2^n)$: problème difficile

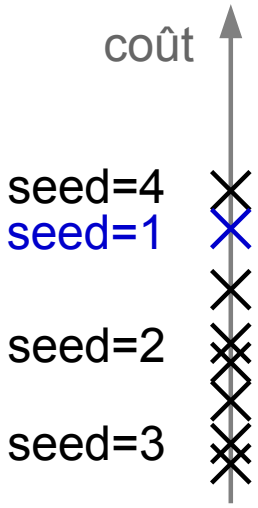
Résolution de problèmes d'ordonnancement complexes

- Ne pas sur-optimiser les modèles
  - La recherche automatique de CP Optimizer repose sur un algorithme stochastique initialisé par une graine (paramètre IloCP::RandomSeed)
  - Pour un même problème, la meilleure solution trouvée dépend de la graine
    Exemple: coût de la solution trouvée à la time-limit:



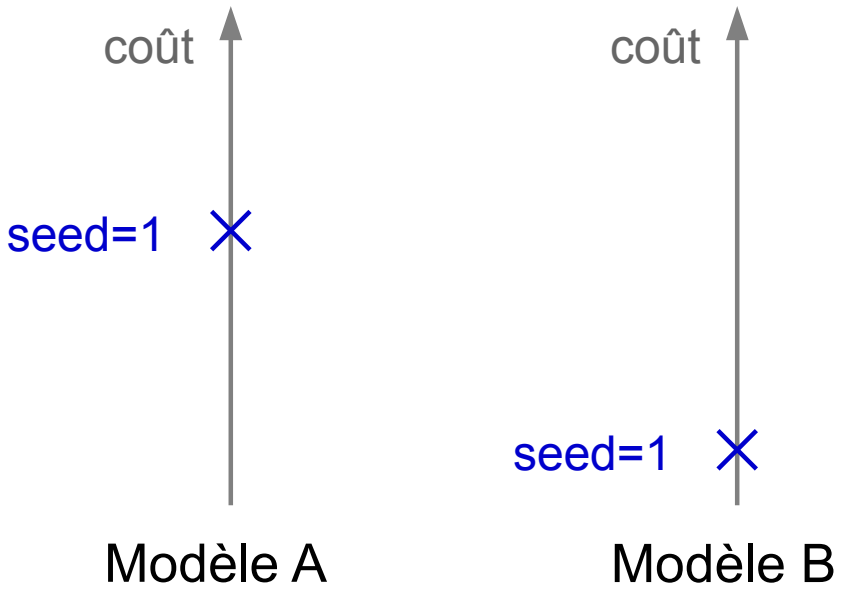
Résolution de problèmes d'ordonnancement complexes

- Ne pas sur-optimiser les modèles
  - La recherche automatique de CP Optimizer repose sur un algorithme stochastique initialisé par une graine (paramètre IloCP::RandomSeed)
  - Pour un même problème, la meilleure solution trouvée dépend de la graine
    Exemple: coût de la solution trouvée à la time-limit:



Résolution de problèmes d'ordonnancement complexes

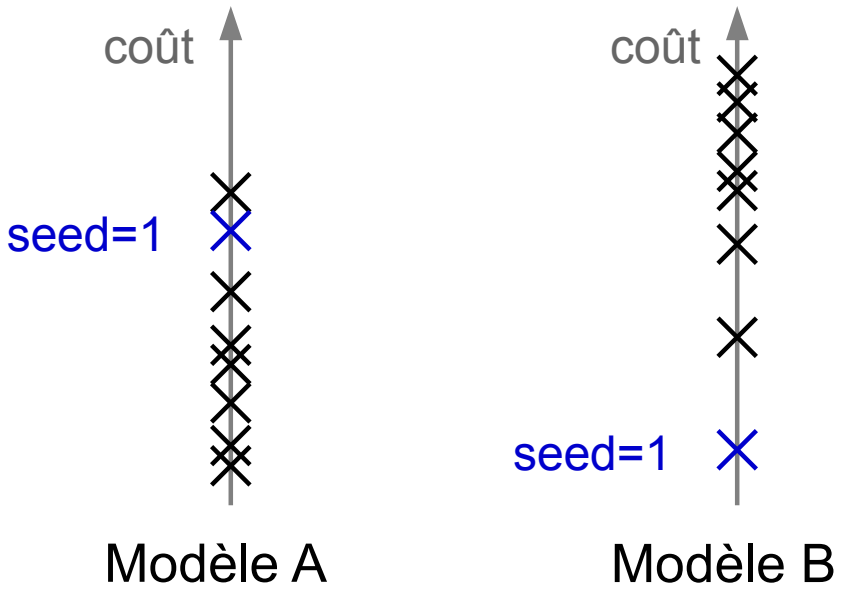
- Ne pas sur-optimiser les modèles
  - La recherche automatique de CP Optimizer repose sur un algorithme stochastique initialisé par une graine (paramètre IloCP::RandomSeed)
  - Pour un même problème, la meilleure solution trouvée dépend de la graine
    Exemple: coût de la solution trouvée à la time-limit:



Le modèle B est-il meilleur que le modèle A?

Résolution de problèmes d'ordonnancement complexes

- Ne pas sur-optimiser les modèles
  - La recherche automatique de CP Optimizer repose sur un algorithme stochastique initialisé par une graine (paramètre IloCP::RandomSeed)
  - Pour un même problème, la meilleure solution trouvée dépend de la graine
    Exemple: coût de la solution trouvée à la time-limit:



Le modèle B est-il meilleur que le modèle A?

Non!

Résolution de problèmes d'ordonnancement complexes

- Optimiser les modèles:
  - Sur des données de taille réelle
  - Sur plusieurs instances (si possible)
  - **Sur plusieurs graines**

Résolution de problèmes d'ordonnancement complexes

- Outils d'analyse:
  - Comment voir si mon modèle est « facile » pour CP Optimizer?
    → **Search log**
  - Pourquoi mon modèle est-il infaisable ?
    → **Conflict refiner**
- Outils de résolution:
  - Comment injecter une solution faisable pour rendre mon modèle « facile » ?
    → **Warmstart**
  - Quels sont les paramètres les plus utiles ?

Résolution de problèmes d'ordonnancement complexes

- Comment voir si mon modèle est « facile » pour CP Optimizer?
  - Lancer la résolution en mode séquentiel (`Workers=1`) en s'arrêtant à la première solution faisable (`SolutionLimit=1`) et observer le nombre de fails dans le log
  - Si #fails est 0 ou de l'ordre de quelques unités → problème facile
  - Sinon, il peut être intéressant de regarder où ont lieu ces fails en affichant chacune des décisions (`LogPeriod=1`) et en stoppant la recherche après quelques fails (ex: `FailLimit=10`)
  - Essayer de reformuler le modèle autour des variables causant les fails pour avoir plus de propagation

Résolution de problèmes d'ordonnancement complexes

- Un modèle « facile » pour CP Optimizer:

```
! ----------------------------------------------------------------------
! Maximization problem - 2,980 variables, 1,689 constraints
! Workers                 = 1
! SolutionLimit           = 1
! Initial process time : 0.01s (0.00s extraction + 0.01s propagation)
!  . Log search space  : 5,769.5 (before), 5,769.5 (after)
!  . Memory usage      : 14.4 MB (before), 17.1 MB (after)
! Using sequential search.
! ----------------------------------------------------------------------
!           Best Branches  Non-fixed            Branch decision
*            791        1 0.10s                 on task#349
! ----------------------------------------------------------------------
! Search terminated by limit, 1 solution found.
! Best objective         : 791
! Number of branches     : 1
! Number of fails        : 0
! Total memory usage     : 21.1 MB (19.0 MB CP Optimizer + 2.1 MB Concert)
! Time spent in solve    : 0.12s (0.12s engine + 0.00s extraction)
! Search speed (br. / s) : 8.0
! ----------------------------------------------------------------------
```

Résolution de problèmes d'ordonnancement complexes

- Pourquoi mon modèle est-il infaisable ?

  → **Conflict Refiner**

Conflict Refiner

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

- Use cases:
  1) Model debugging (errors in model)
  2) Data debugging (inconsistent data)
  3) The model and data are correct, but the associated data represents a real-world conflict in the system being modeled
  4) You create an infeasible model to test properties of (or extract information about) a similar model

Conflict Refiner

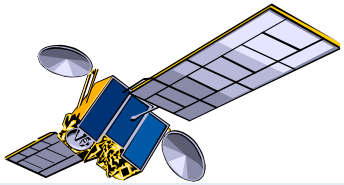
- Objective: Identify a reason for an inconsistency by providing a minimal infeasible subset of constraints for an infeasible model

- Use cases:
  1) Model debugging (errors in model)
  2) Data debugging (inconsistent data)
  3) The model and data are correct, but the associated data represents a real-world conflict in the system being modeled
  4) You create an infeasible model to test properties of (or extract information about) a similar model

Conflict Refiner example: satellite scheduling problem

- Satellite communication 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

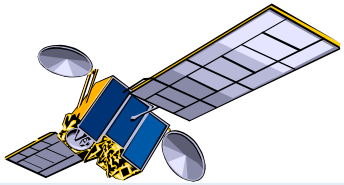
[1] Kramer & al.: Understanding Performance Trade-offs in Algorithms for Solving Oversubscribed Scheduling.

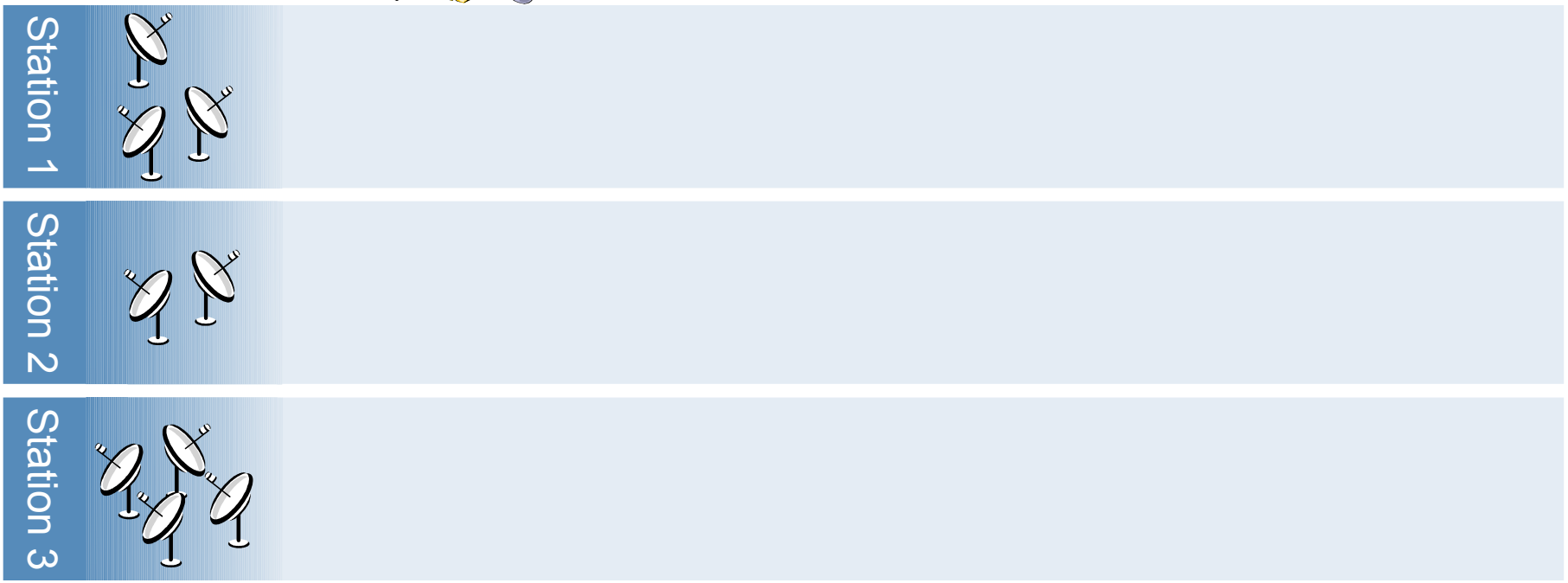


Station 1

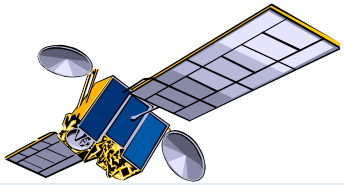
Station 2

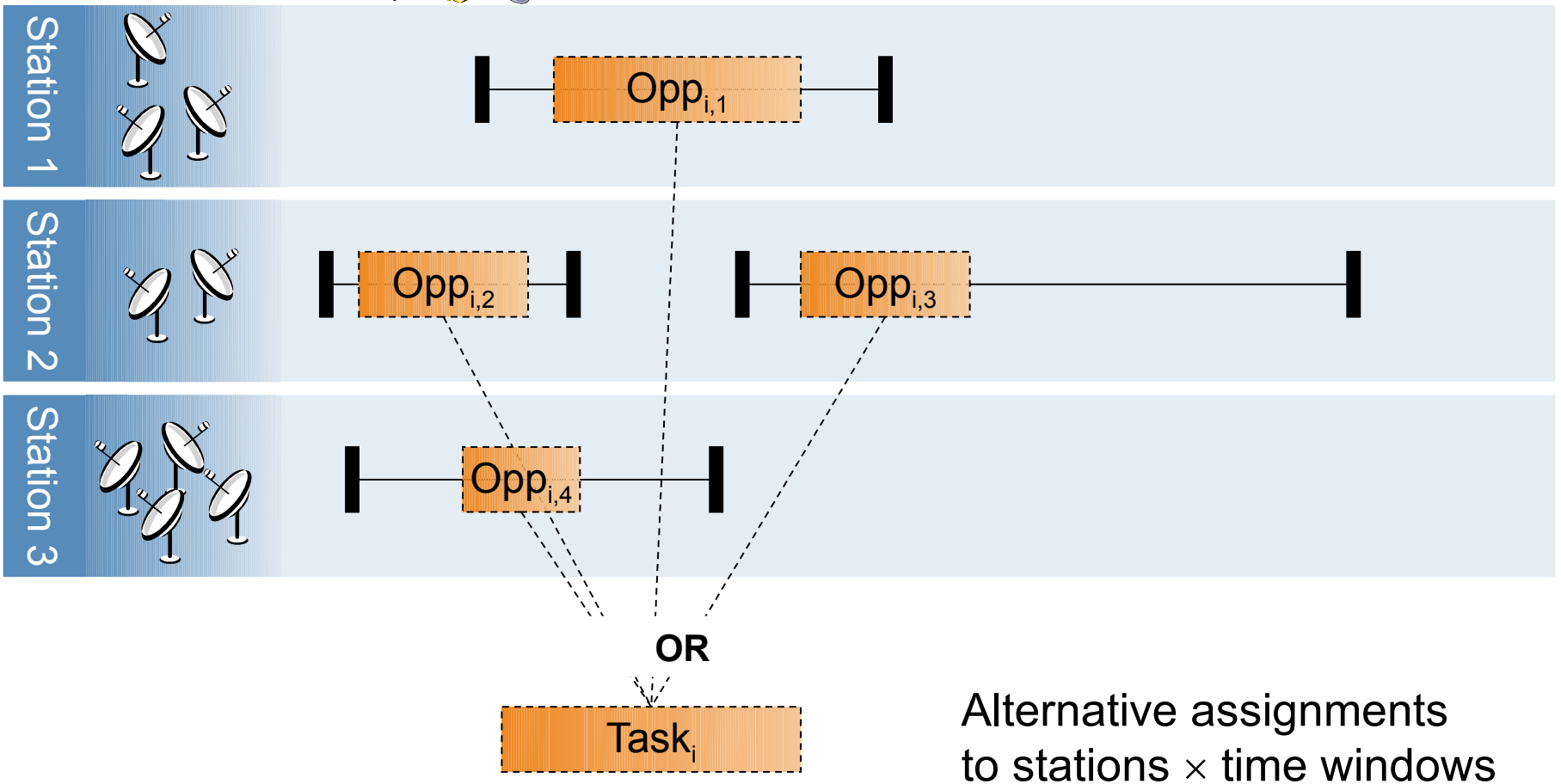
Station 3



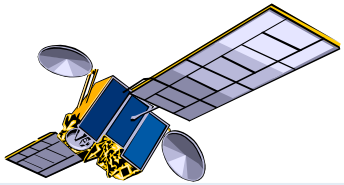


$Task_i$ Communication requests





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

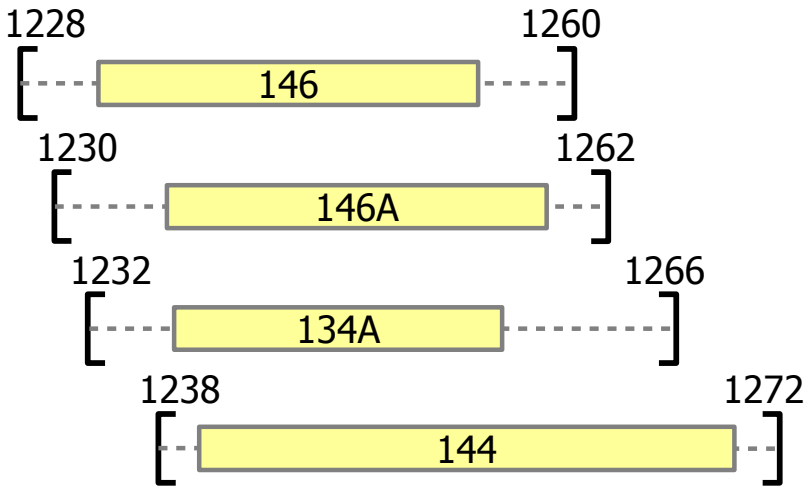
Conflict Refiner example: running model 2

- 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

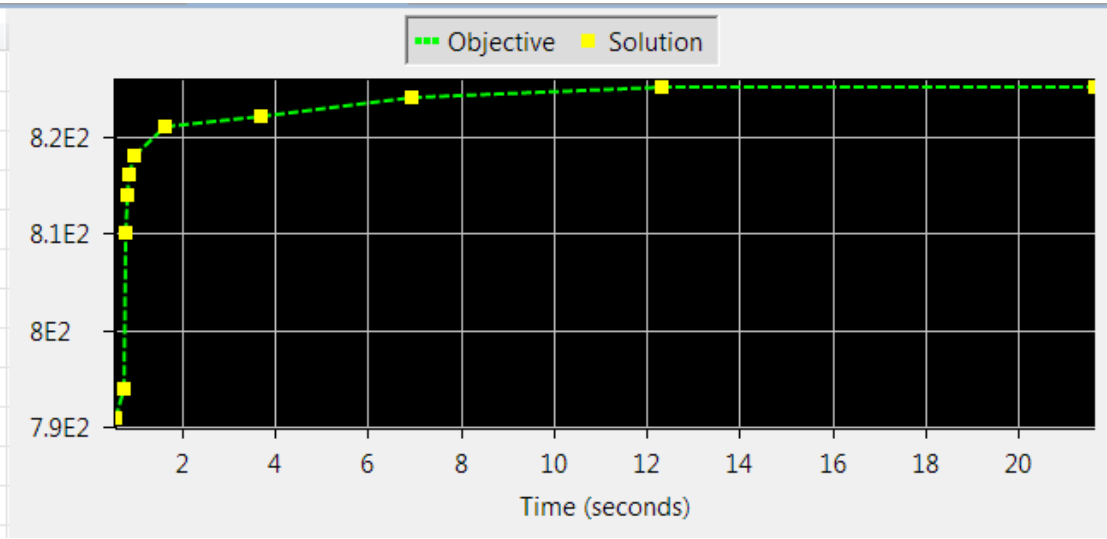
```
1  using CP;
2
3  tuple Station {
4    string name; // Ground station name
5    int id;      // Ground station identifier
6    int cap;     // Number of available antennas
7  }
8
9  tuple Opportunity {
10   string task; // Task
11   int station; // Ground station
12   int smin;    // Start of visibility 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 }
```

Conflict Refiner example: running model 3

Statistic	Value
CP	
Constraints	1689
Variables	2980
Memory usage	61007896
Choice points	620122
Number of solutions	10
Number of branches	713746
Number of fails	95284
Scheduling	
Number of intervals	2980
Objective	825



- Solution with 825 tasks executed for a total of 838 candidates (98.4%)

Résolution de problèmes d'ordonnancement complexes

- Comment injecter une solution faisable pour rendre mon modèle « facile » ?

  → **Warmstart**

Warmstart

- Objective: Start search from a known (possibly incomplete) solution given by the user in order to further improve it or to help to guide the engine towards a first solution

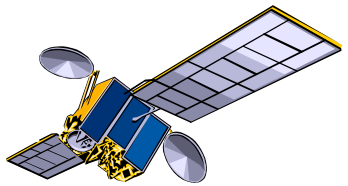
- Use cases:
  1) Restart an interrupted search with the current incumbent
  2) Start from an initial solution found by an available heuristic
  3) Goal programming for multi-objective problems
  4) When finding an initial solution is hard, solve an initial problem that minimizes constraint violation and start from its solution
  5) Successively solving similar problems (e.g. dynamic scheduling)
  6) Hierarchical problem solving (e.g. planning → scheduling)

Warmstart

- Objective: Start search from a known (possibly incomplete) solution given by the user in order to further improve it or to help to guide the engine towards a first solution

- Use cases:
  1) Restart an interrupted search with the current incumbent
  2) Start from an initial solution found by an available heuristic
  3) Goal programming for multi-objective problems
  4) When finding an initial solution is hard, solve an initial problem that minimizes constraint violation and start from its solution
  5) Successively solving similar problems (e.g. dynamic scheduling)
  6) Hierarchical problem solving (e.g. planning → scheduling)

Warmstart example: satellite scheduling problem



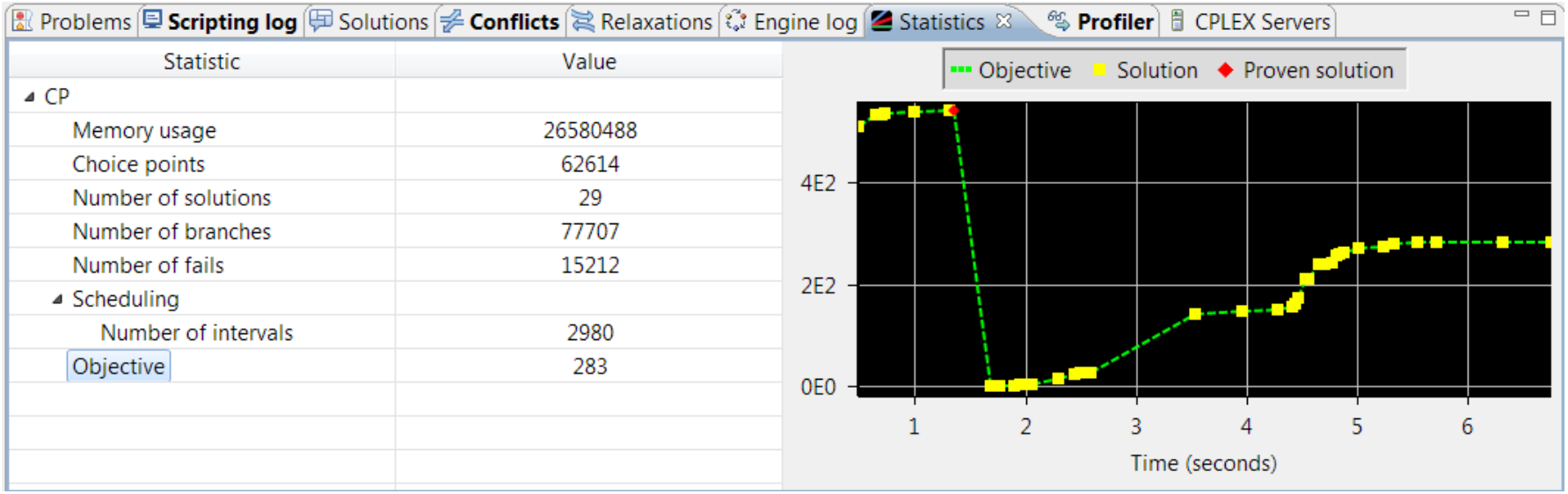
- Satellite communication scheduling problem [1]
- *n* communication requests for Earth orbiting satellites must be scheduled on a total of 32 antennas spread across 13 ground-based tracking stations
- In the instances, *n* ranges from to 400 to 1300
- Tasks have priorities: first maximize the number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

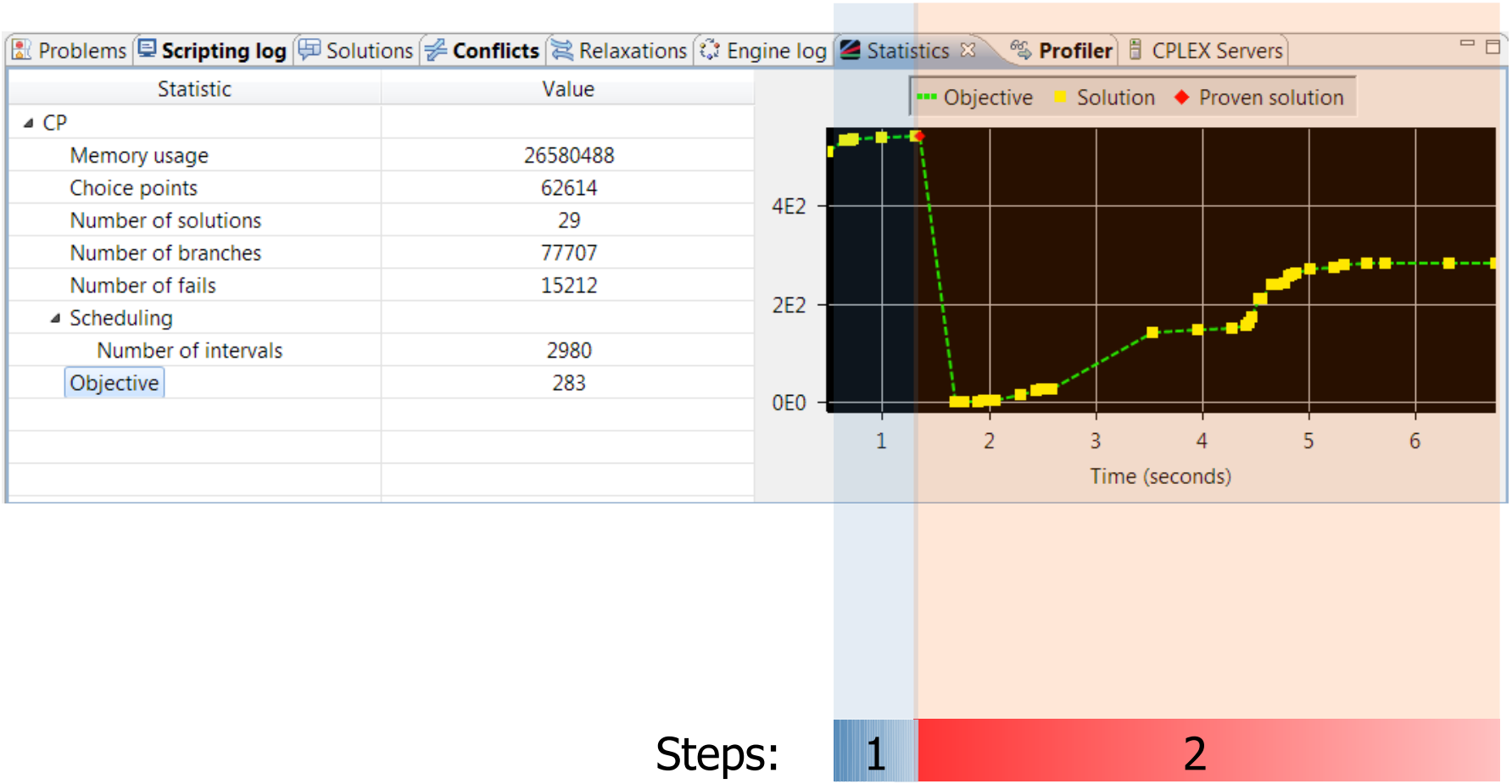
Warmstart example: satellite scheduling problem

```
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();
```

Warmstart example: satellite scheduling problem



Warmstart example: satellite scheduling problem



Résolution de problèmes d'ordonnancement complexes

- Quels sont les paramètres les plus utiles ?
  - Analyse et mise au point du modèle:
    - Workers=1, LogPeriod=1, SolutionLimit=1, FailLimit=1
    - RandomSeed
  - TimeLimit
  - Workers
  - TemporalRelaxation=On/Off
  - SearchType=Restart/MultiPoint

Résolution de problèmes d'ordonnancement complexes

- Quelques éléments de méthodologie:
  - Exploiter la richesse du modèle d'ordonnancement
  - Ne pas surexploiter la richesse du modèle d'ordonnancement
  - Privilégier les modèles *faciles*
  - Travailler le plus tôt possible avec des données de taille réelle
  - Ne pas sur-optimiser les modèles
- Outils d'analyse:
  - Comment voir si mon modèle est « facile » pour CP Optimizer?
    → **Search log**
  - Pourquoi mon modèle est-il infaisable ?
    → **Conflict refiner**
- Outils de résolution:
  - Comment injecter une solution faisable pour rendre mon modèle « facile » ?
    → **Warmstart**
  - Paramètres les plus utiles:
    → **Limits, Workers, TemporalRelaxation, SearchType**