

IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems

May 30, 2009

Philippe Laborie
Principal Scientist
laborie@fr.ibm.com

What is IBM ILOG CP Optimizer?



- A component of IBM ILOG CPLEX Optimization Studio
- A Constraint Programming engine for combinatorial problems (including scheduling problems)
- Implements a Model & Run paradigm
 - Model: **Concise and Expressive modeling language**
 - Run: **Powerful automatic search procedure**
- Available through the following interfaces:
 - OPL
 - C++ (native interface)
 - Java, .NET (wrapping of the C++ engine)

- CP Optimizer provides some specific decision variables and constraints for modeling and solving detailed scheduling problems

Modeling Language [1,2]

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

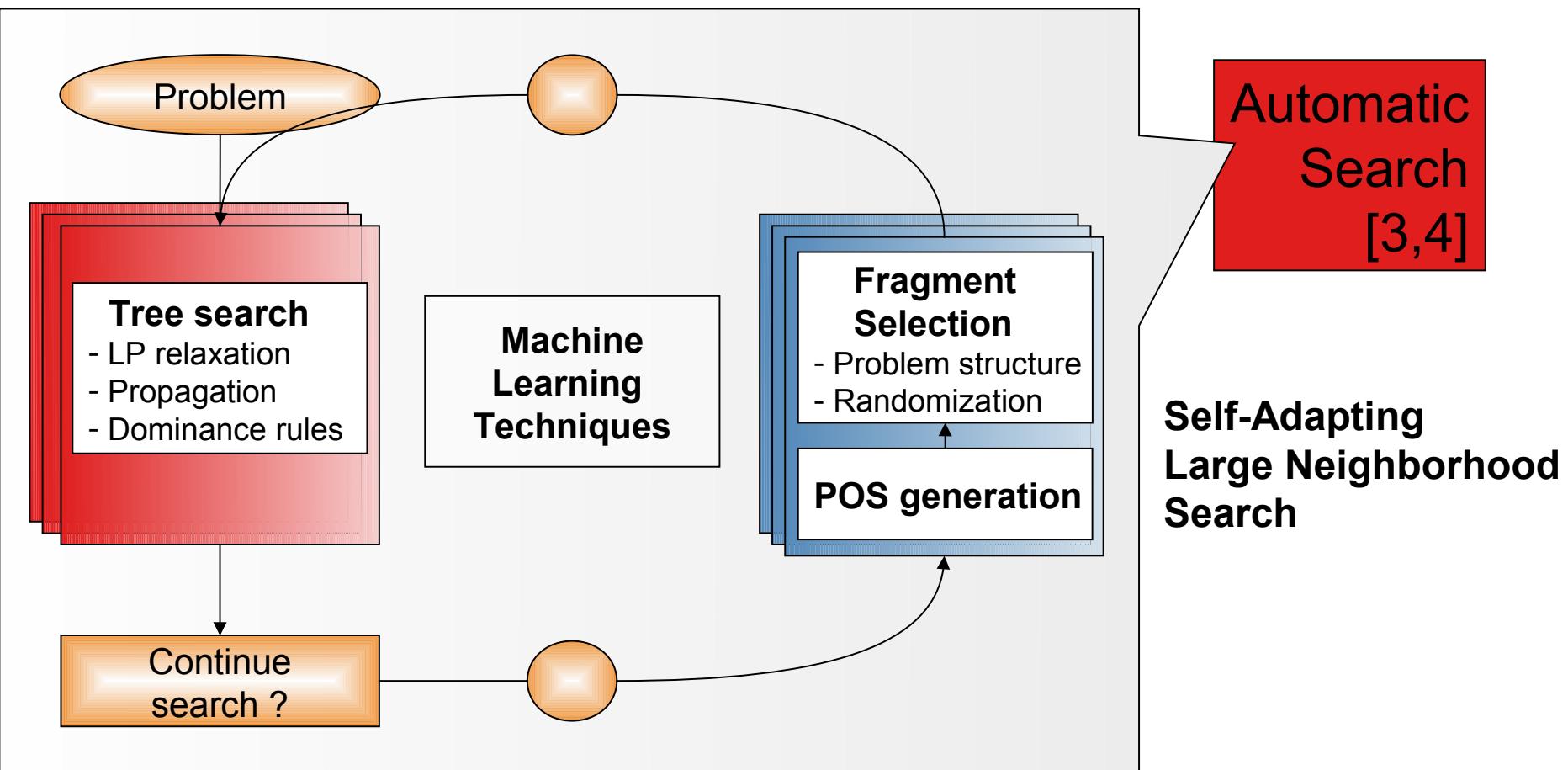
$$\text{Domain}(a) \subseteq \{\perp\} \cup \{ [s,e) \mid s, e \in \mathbb{Z}, s \leq e \}$$


Absent interval Interval of integers

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

[1] Reasoning with Conditional Time-intervals. FLAIRS-08.

[2] Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. FLAIRS-09.



Automatic
Search
[3,4]

Self-Adapting
Large Neighborhood
Search

[3] Randomized Large Neighborhood Search for Cumulative Scheduling. ICAPS-05.

[4] Self-Adapting Large Neighborhood Search: Application to Single-mode Scheduling Problems. MISTA-07.

[1] Reasoning with Conditional Time-intervals. FLAIRS-08.

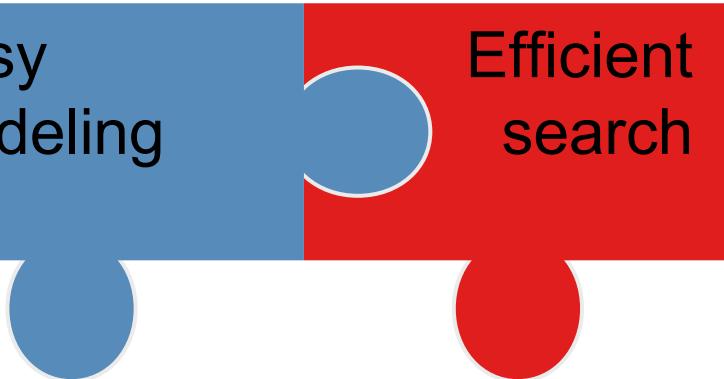
[2] Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. FLAIRS-09.

Modeling
Language
[1,2]

Easy
modeling

Efficient
search

Automatic
Search
[3,4]



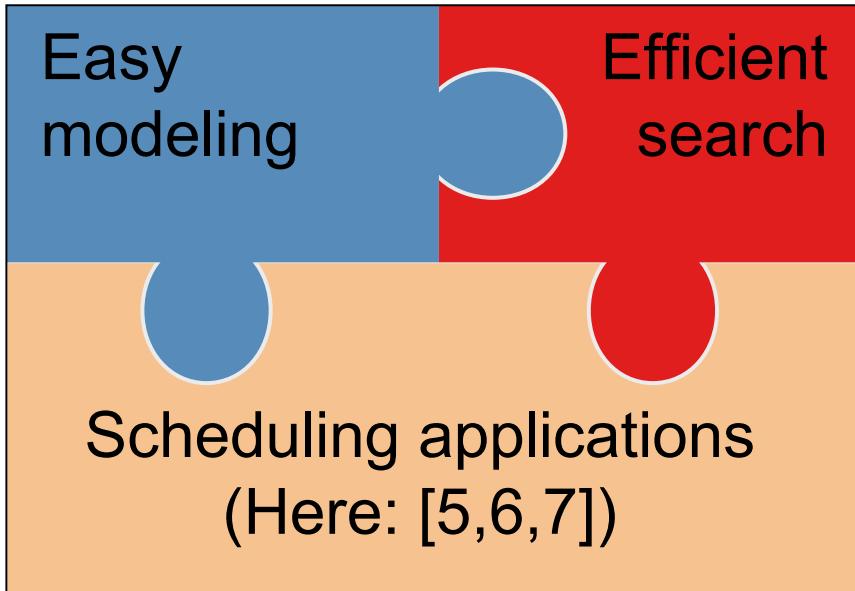
[3] Randomized Large Neighborhood Search for Cumulative Scheduling. ICAPS-05.

[4] Self-Adapting Large Neighborhood Search: Application to Single-mode Scheduling Problems. MISTA-07.

[1] Reasoning with Conditional Time-intervals. FLAIRS-08.

[2] Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. FLAIRS-09.

Modeling
Language
[1,2]

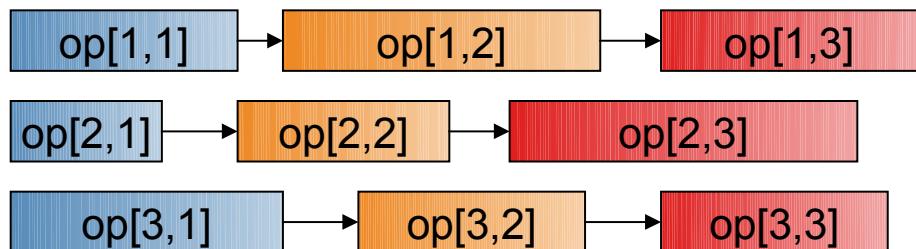


Automatic
Search
[3,4]

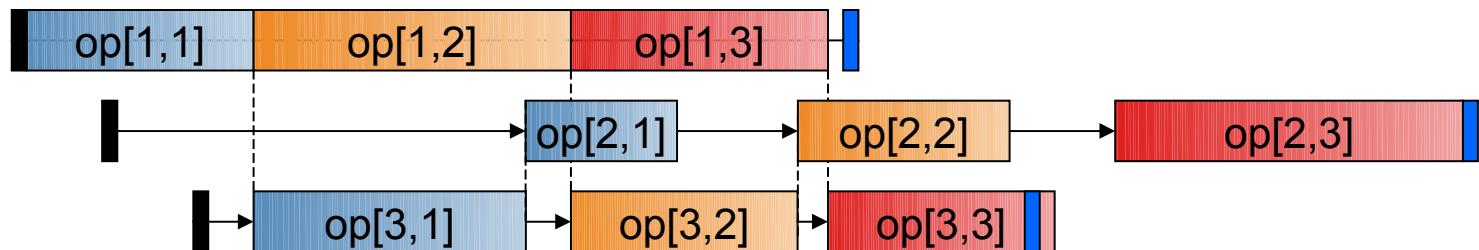
- [5] Danna, Perron: Structured vs. Unstructured Large Neighborhood Search.
 - [6] Kramer & al.: Understanding Performance Trade-offs in Algorithms for Solving Oversubscribed Scheduling.
 - [7] Refanidis: Managing Personal Tasks with Time Constraints and Preferences.
-
- [3] Randomized Large Neighborhood Search for Cumulative Scheduling. ICAPS-05.
 - [4] Self-Adapting Large Neighborhood Search: Application to Single-mode Scheduling Problems. MISTA-07.
-
- [1] Reasoning with Conditional Time-intervals. FLAIRS-08.
 - [2] Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. FLAIRS-09.

Problem #1: Flow-shop with Earliness/Tardiness

- Classical Flow-Shop Scheduling problem:
 - n jobs, m machines



- Job release dates (|), due dates (|) and weight



Problem #1: Flow-shop with Earliness/Tardiness

```

1 using CP;
2 int n = ....;
3 int m = ....;
4 int rd[1..n] = ....;
5 int dd[1..n] = ....;
6 float w[1..n] = ....;
7 int pt[1..n][1..m] = ....;
8 float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9 dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endof(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13     forall(i in 1..n) {
14         rd[i] <= startOf(op[i][1]);
15         forall(j in 1..m-1)
16             endBeforeStart(op[i][j],op[i][j+1]);
17     }
18     forall(j in 1..m)
19         noOverlap(all(i in 1..n) op[i][j]);
20 }
```

- The model is using CP Optimizer engine

Problem #1: Flow-shop with Earliness/Tardiness

```

1 using CP;
2 int n = ....;
3 int m = ....;
4 int rd[1..n] = ....;
5 int dd[1..n] = ....;
6 float w[1..n] = ....;
7 int pt[1..n][1..m] = ....;
8 float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9 dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endof(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13     forall(i in 1..n) {
14         rd[i] <= startOf(op[i][1]);
15         forall(j in 1..m-1)
16             endBeforeStart(op[i][j],op[i][j+1]);
17     }
18     forall(j in 1..m)
19         noOverlap(all(i in 1..n) op[i][j]);
20 }
```

- Data reading and computation:
 - Problem size (n, m)
 - Job release date (rd), due-date (dd), weight (w)
 - Operation processing time (pt)
 - Normalization factor (W)

Problem #1: Flow-shop with Earliness/Tardiness

```

1 using CP;
2 int n = ....;
3 int m = ....;
4 int rd[1..n] = ....;
5 int dd[1..n] = ....;
6 float w[1..n] = ....;
7 int pt[1..n][1..m] = ....;
8 float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9 dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endof(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13     forall(i in 1..n) {
14         rd[i] <= startOf(op[i][1]);
15         forall(j in 1..m-1)
16             endBeforeStart(op[i][j],op[i][j+1]);
17     }
18     forall(j in 1..m)
19         noOverlap(all(i in 1..n) op[i][j]);
20 }
```

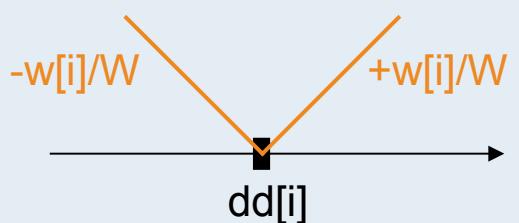
- Decision variables and expressions:
 - Operations: 2D array of **interval variables**
 - Jobs end time: 1D array of **integer expressions**

Problem #1: Flow-shop with Earliness/Tardiness

```

1 using CP;
2 int n = ....;
3 int m = ....;
4 int rd[1..n] = ....;
5 int dd[1..n] = ....;
6 float w[1..n] = ....;
7 int pt[1..n][1..m] = ....;
8 float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9 dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endof(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13     forall(i in 1..n) {
14         rd[i] <= startOf(op[i][1]);
15         forall(j in 1..m-1)
16             endBeforeStart(op[i][j],op[i][j+1]);
17     }
18     forall(j in 1..m)
19         noOverlap(all(i in 1..n) op[i][j]);
20 }
```

- **Objective:**
 - Weighted sum of earliness/tardiness costs



Problem #1: Flow-shop with Earliness/Tardiness

```

1 using CP;
2 int n = ....;
3 int m = ....;
4 int rd[1..n] = ....;
5 int dd[1..n] = ....;
6 float w[1..n] = ....;
7 int pt[1..n][1..m] = ....;
8 float W = sum(i in 1..n) (w[i] * sum(j in 1..m) pt[i][j]);
9 dvar interval op[i in 1..n][j in 1..m] size pt[i][j];
10 dexpr int C[i in 1..n] = endof(op[i][m]);
11 minimize sum(i in 1..n) w[i]*abs(C[i]-dd[i])/W;
12 subject to {
13     forall(i in 1..n) {
14         rd[i] <= startOf(op[i][1]);
15         forall(j in 1..m-1)
16             endBeforeStart(op[i][j],op[i][j+1]);
17     }
18     forall(j in 1..m)
19         noOverlap(all(i in 1..n) op[i][j]);
20 }
```

Constraints

- **Constraints:**
 - Release dates of job i (using **startOf** integer expression)
 - **Precedence constraints** between operations of job i
 - **No-overlap** of operations on the same machine j

Problem #1: Flow-shop with Earliness/Tardiness

- Experimental results

Problem	GA-best	S-LNS-best	<i>CPO</i>	Problem	GA-best	S-LNS-best	<i>CPO</i>
jb1	0.474	0.191	<i>0.191</i>	ljb1	0.279	0.215	<i>0.215</i>
jb2	0.499	0.137	<i>0.137</i>	ljb2	0.598	0.508	<i>0.509</i>
jb4	0.619	0.568	<i>0.568</i>	ljb7	0.246	0.110	<i>0.137</i>
jb9	0.369	0.333	<i>0.334</i>	ljb9	0.739	1.015	<i>0.744</i>
jb11	0.262	0.213	<i>0.213</i>	ljb10	0.512	0.525	<i>0.549</i>
jb12	0.246	0.190	<i>0.190</i>	ljb12	0.399	0.605	<i>0.518</i>

Table 1. Results for Flow-shop Scheduling with Earliness and Tardiness Costs

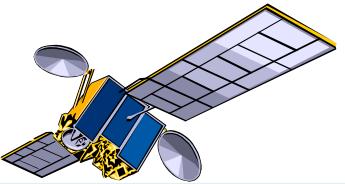
- Average improvement:
 - Compared to GA: 25%
 - Compared to LNS: 1.7%

[5] Danna, Perron: Structured vs. Unstructured Large Neighborhood Search.

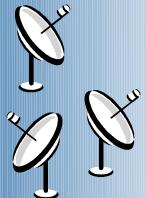
Problem #2: Oversubscribed Scheduling

- USAF Satellite Control Network scheduling problem [6]
- A set of n communication requests for Earth orbiting satellites must be scheduled on a total of 32 antennas spread across 13 ground-based tracking stations.
- Objective is to maximize the number of satisfied requests
- In the instances, n ranges from to 400 to 1300

Problem #2: Oversubscribed Scheduling



Station 1



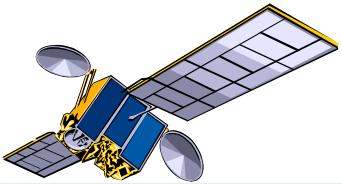
Station 2



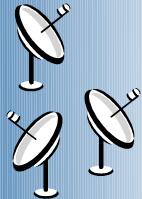
Station 3



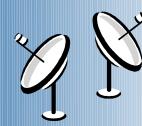
Problem #2: Oversubscribed Scheduling



Station 1



Station 2

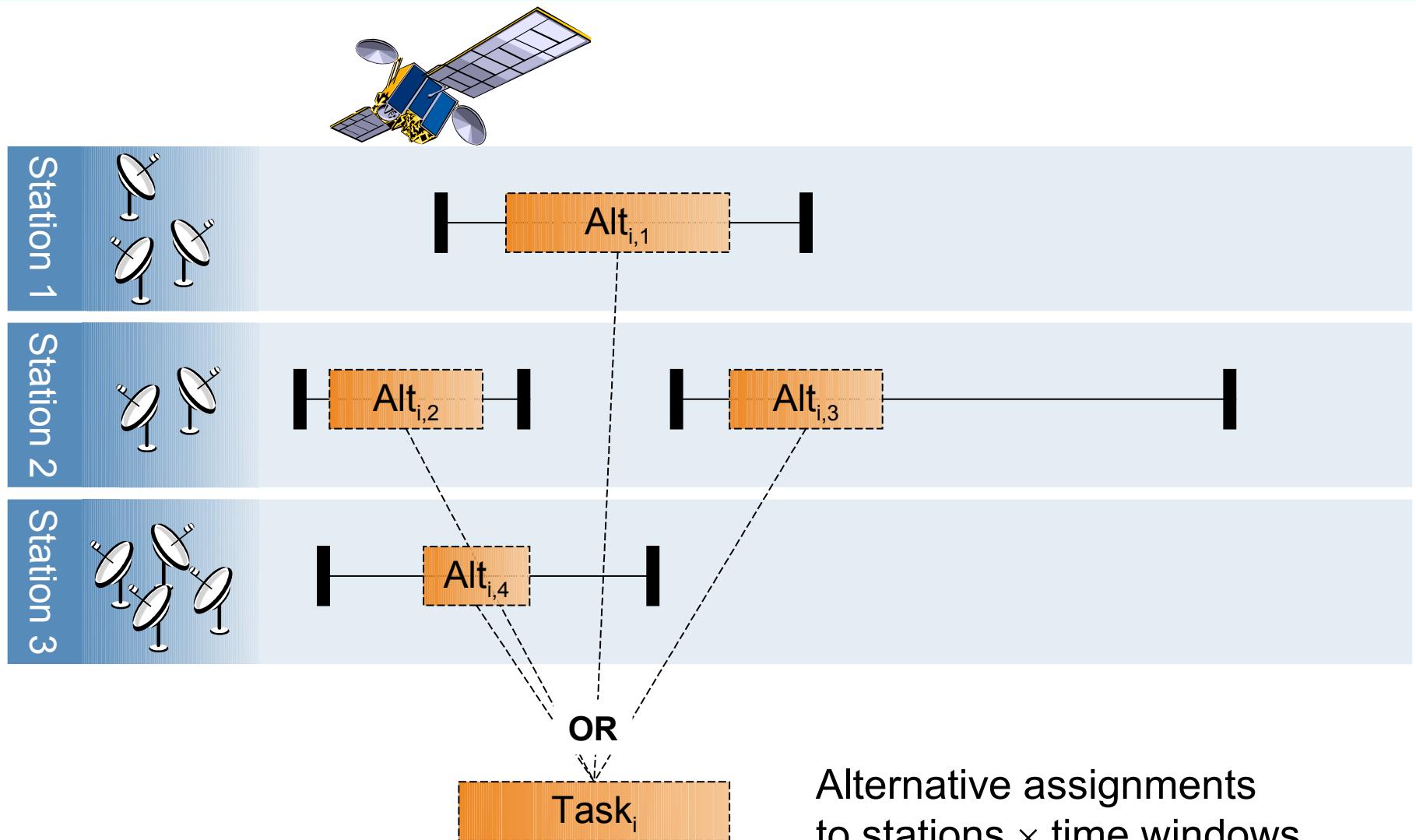


Station 3

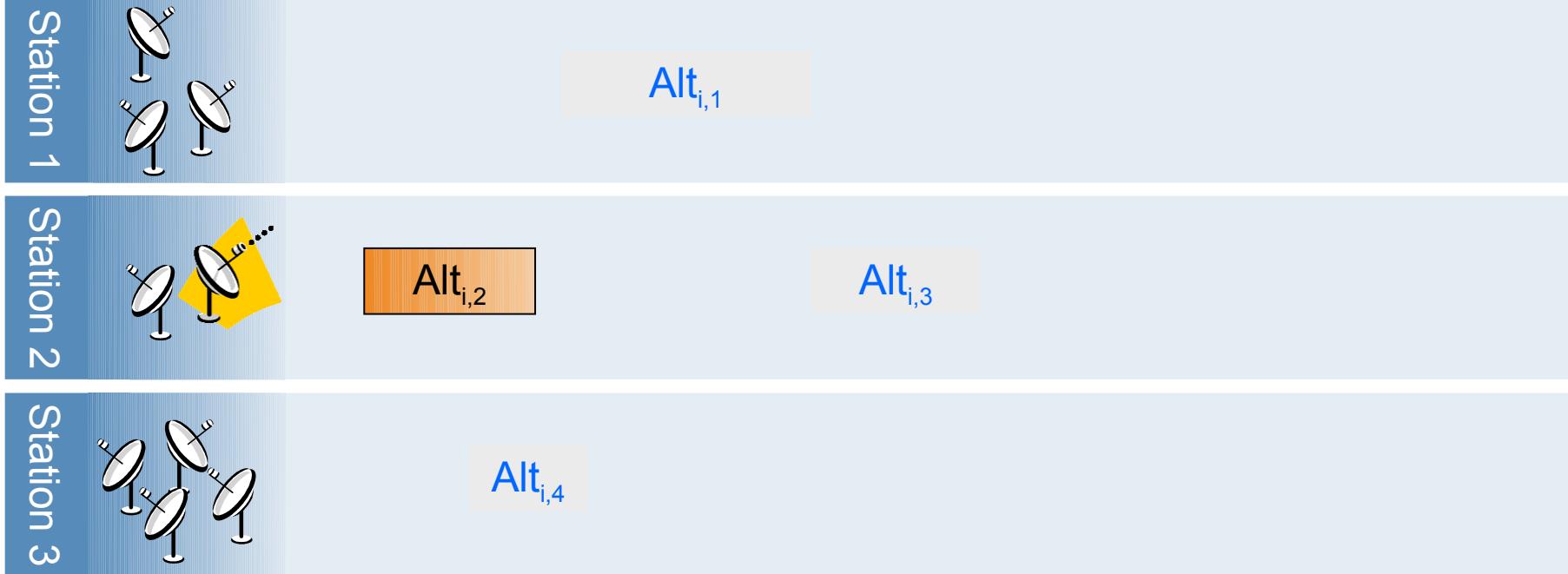
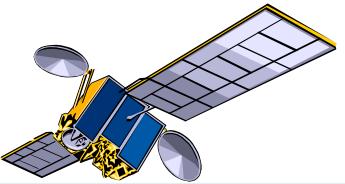


Communication requests

Problem #2: Oversubscribed Scheduling



Problem #2: Oversubscribed Scheduling



Selected alternative will use 1 antenna for communication with the satellite

Problem #2: Oversubscribed Scheduling

```

1 using CP;
2 tuple Station { string name; int id; int cap; }
3 tuple Alternative { string task; int station; int smin; int dur; int emax; }
4 {Station} Stations = ...;
5 {Alternative} Alternatives = ...;
6 {string} Tasks = { a.task | a in Alternatives };
7 dvar interval task[t in Tasks] optional;
8 dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
9 maximize sum(t in Tasks) presenceOf(task[t]);
10 subject to {
11     forall(t in Tasks)
12         alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13     forall(s in Stations)
14         sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15 }
```

- Data reading and computation:
 - Tuple sets of Stations and Alternative assignments
 - Tasks: set of tasks of the problem

Problem #2: Oversubscribed Scheduling

```

1 using CP;
2 tuple Station { string name; int id; int cap; }
3 tuple Alternative { string task; int station; int smin; int dur; int emax; }
4 {Station} Stations = ...;
5 {Alternative} Alternatives = ...;
6 {string} Tasks = { a.task | a in Alternatives };
7 dvar interval task[t in Tasks] optional;
8 dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
9 maximize sum(t in Tasks) presenceOf(task[t]);
10 subject to {
11     forall(t in Tasks)
12         alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13     forall(s in Stations)
14         sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15 }
```

- Decision variables:
 - Tasks: array of **optional** interval variables
 - Alternative assignments: array of **optional** interval variables,
each possible assignment is defined with a specific time-window ([smin,emax]) and a size

Problem #2: Oversubscribed Scheduling

```

1 using CP;
2 tuple Station { string name; int id; int cap; }
3 tuple Alternative { string task; int station; int smin; int dur; int emax; }
4 {Station} Stations = ...;
5 {Alternative} Alternatives = ...;
6 {string} Tasks = { a.task | a in Alternatives };
7 dvar interval task[t in Tasks] optional;
8 dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
9 maximize sum(t in Tasks) presenceOf(task[t]);
10 subject to {
11     forall(t in Tasks)
12         alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13     forall(s in Stations)
14         sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15 }
```

Objective

- **Objective:**
 - Maximize number of executed tasks (modeled with a sum of **presenceOf** constraints)

Problem #2: Oversubscribed Scheduling

```

1 using CP;
2 tuple Station { string name; int id; int cap; }
3 tuple Alternative { string task; int station; int smin; int dur; int emax; }
4 {Station} Stations = ...;
5 {Alternative} Alternatives = ...;
6 {string} Tasks = { a.task | a in Alternatives };
7 dvar interval task[t in Tasks] optional;
8 dvar interval alt[a in Alternatives] optional in a.smin..a.emax size a.dur;
9 maximize sum(t in Tasks) presenceOf(task[t]);
10 subject to {
11     forall(t in Tasks)
12         alternative(task[t], all(a in Alternatives: a.task==t) alt[a]);
13     forall(s in Stations)
14         sum(a in Alternatives: a.station==s.id) pulse(alt[a],1) <= s.cap;
15 }
```

- Constraints:
 - Alternative assignments for a given task t using an **alternative constraint**
 - Maximal capacity of stations (number of antennas) using a constrained **cumul function**

■ Experimental Results

Problem set	TS	SWO	CPO	Problem set	TS	SWO	CPO
1.1	30.44	26.60	27.50	4.1	3.20	2.00	1.96
1.2	114.02	104.72	98.10	4.2	13.34	7.90	7.48
1.3	87.92	84.52	86.04	4.3	16.60	12.46	9.68
2.1	11.46	7.80	7.84	5.1	3.90	3.80	3.76
2.2	45.54	34.26	30.64	5.2	32.98	31.98	31.72
2.3	33.96	31.18	32.14	5.3	46.18	45.22	44.34
3.1	2.64	2.32	2.28	6.1	1.56	1.28	1.24
3.2	15.50	12.82	11.82	6.2	11.62	9.56	8.92
3.3	32.10	28.58	24.00	6.3	25.28	22.60	19.48

Table 2. Results for Satellite Scheduling

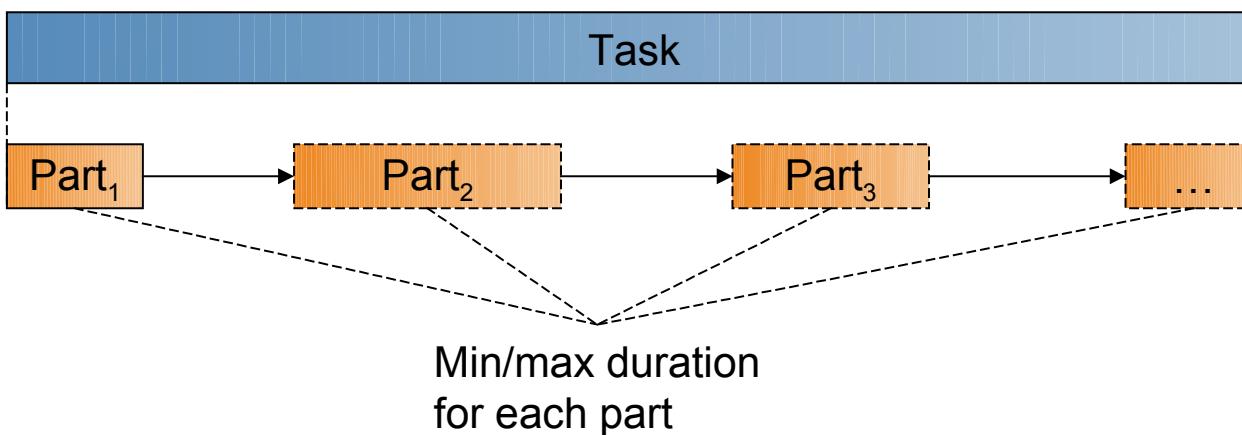
- In average, compared with SWO, the number of unscheduled tasks is decreased by 5.3%

Problem #3: Personal Tasks Scheduling

- Personal tasks scheduling [7]
- Schedule a personal agenda composed of n tasks
- Integration with Google Calendar available online:
<http://selfplanner.uom.gr/>

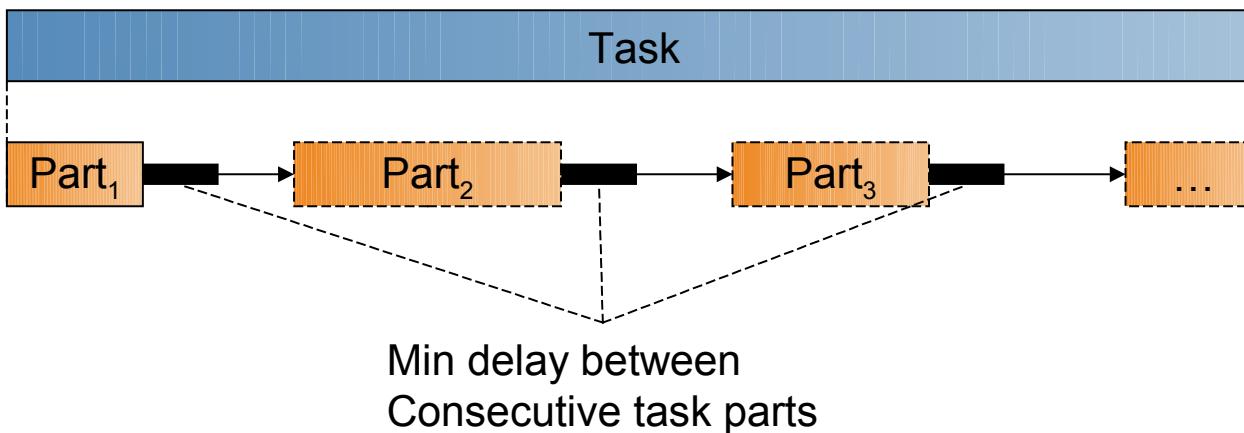
Problem #3: Personal Tasks Scheduling

- Tasks are preemptive
 - A task specifies a fixed total processing time
 - It can be split into one or several parts
 - Min/max value for the duration of each individual part
 - Minimal delay between consecutive parts of the same task



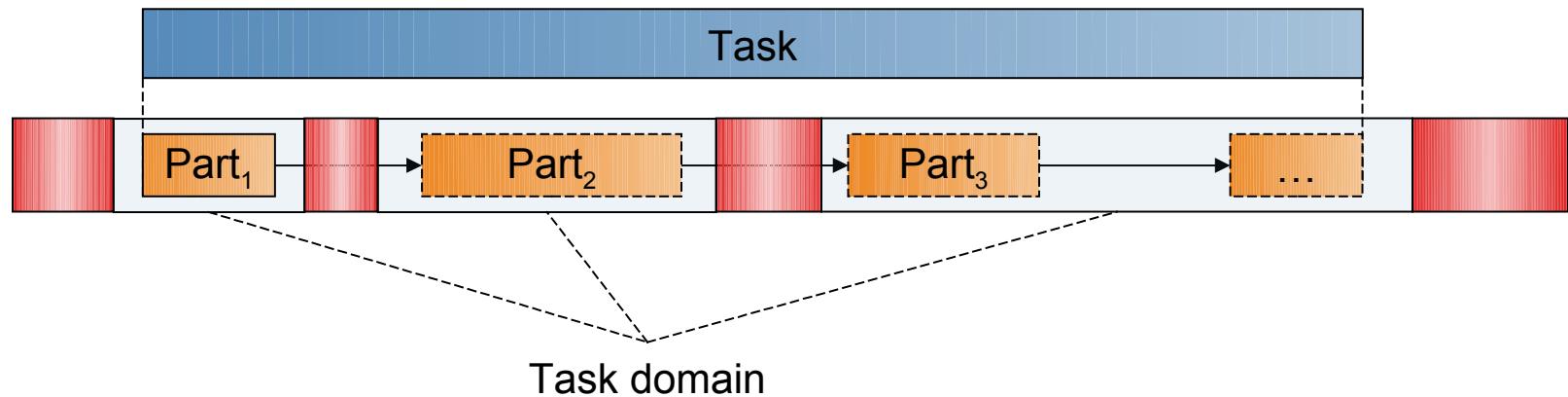
Problem #3: Personal Tasks Scheduling

- Tasks are preemptive
 - A task specifies a fixed total processing time
 - It can be split into one or several parts
 - Min/max value for the duration of each individual part
 - Minimal delay between consecutive parts of the same task



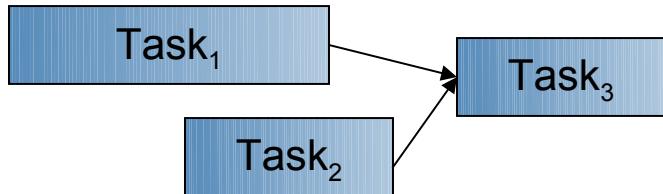
Problem #3: Personal Tasks Scheduling

- Each task specifies a set of time-windows where it can be executed

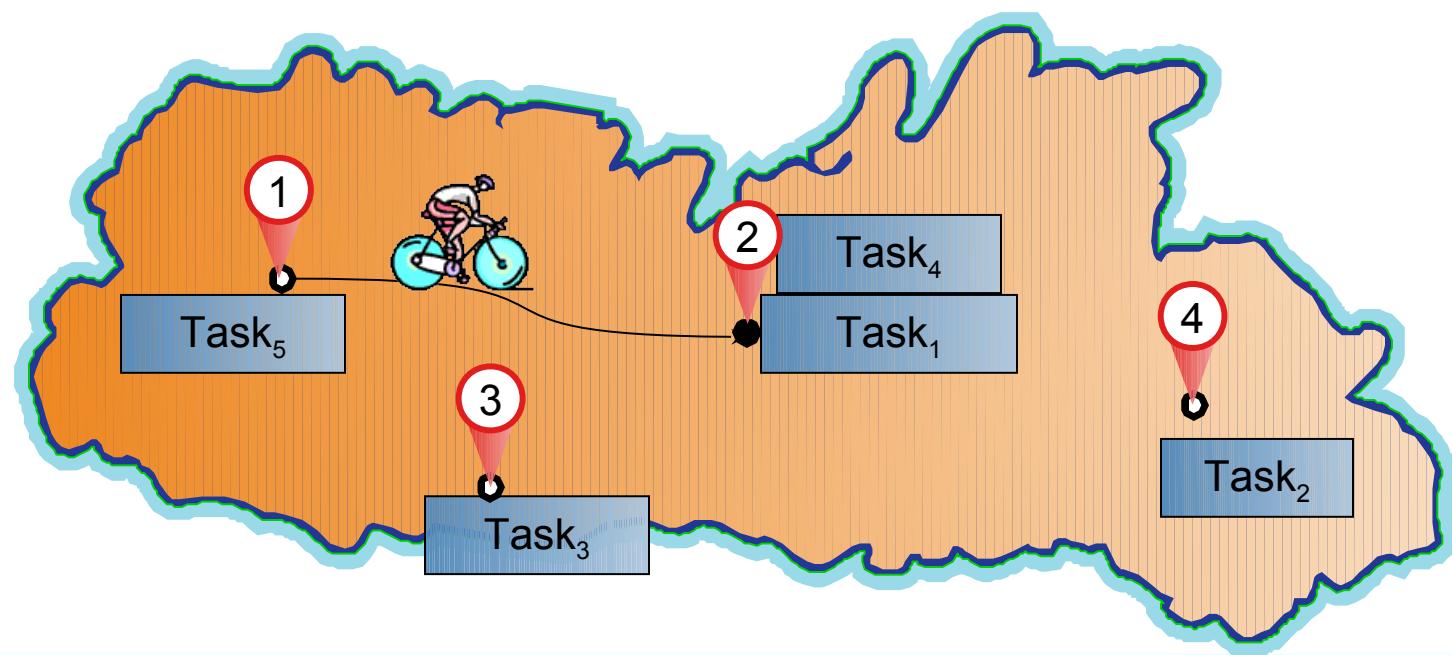


Problem #3: Personal Tasks Scheduling

- Precedence constraints

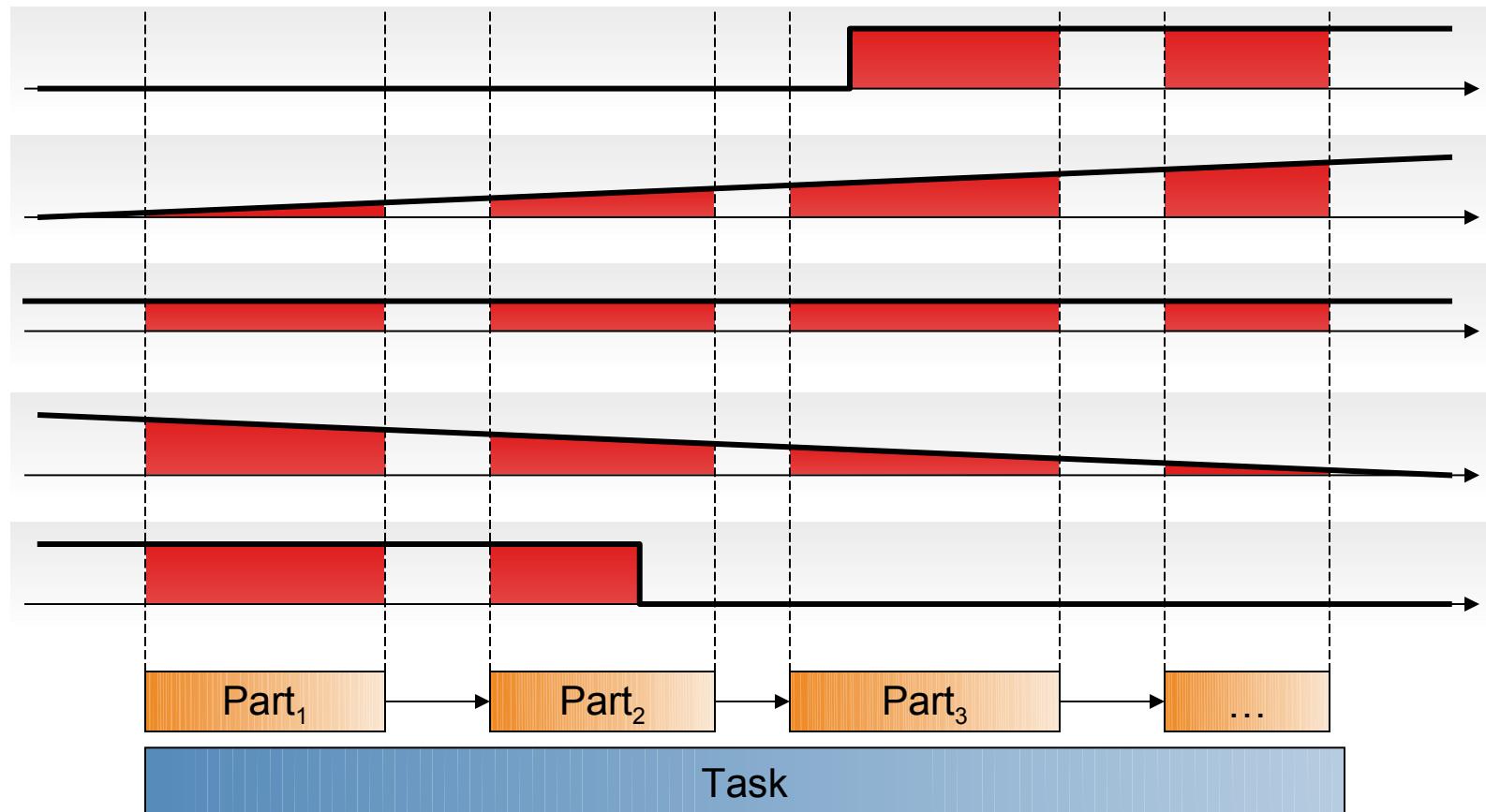


- Locations, distances and transition times



Problem #3: Personal Tasks Scheduling

- Objective function: maximize task satisfaction
 - 5 types of task-dependent preference functions:



Problem #3: Personal Tasks Scheduling

```

1 using CP;
2 tuple Task { key int id; int loc; int dur; int smin; int smax; int dmin;
3                 int f; int date; {int} ds; {int} de; }
4 {Task} Tasks = ....;
5 tuple Distance { int loc1; int loc2; int dist; };
6 {Distance} Distances = ....;
7 tuple Ordering { int pred; int succ; };
8 {Ordering} Orderings = ....;
9 tuple Part { Task task; int id; }
10 {Part} Parts = { <t,i> | t in Tasks, i in 1 .. t.dur div t.smin };
11 tuple Step { int x; int y; }
12 sorted {Step} Steps[t in Tasks] = {<x,0> | x in t.ds} union {<x,1> | x in t.de};
13 stepFunction holes[t in Tasks] = stepwise(s in Steps[t]) {s.y -> s.x; 0};
14 dvar interval tasks[t in Tasks] in 0..500;
15 dvar interval a[p in Parts] optional size p.task.smin..p.task.smax;
16 dvar sequence seq in all(p in Parts) a[p] types all(p in Parts) p.task.loc;

```

- Data reading:
 - Tasks characteristics

Problem #3: Personal Tasks Scheduling

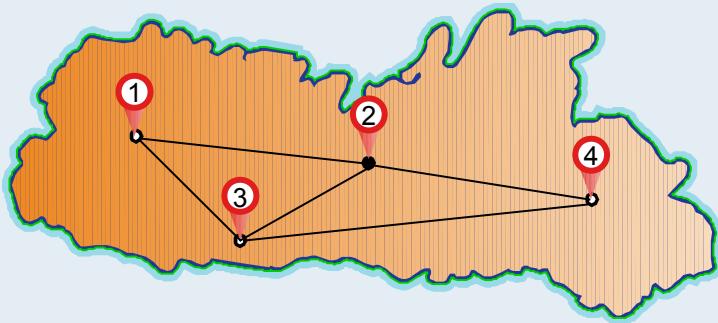
```

1 using CP;
2 tuple Task { key int id; int loc; int dur; int smin; int smax; int dmin;
3                 int f; int date; {int} ds; {int} de; }
4 {Task} Tasks = ....;
5 tuple Distance { int loc1; int loc2; int dist; };
6 {Distance} Distances = ....;
7 tuple Ordering { int pred; int succ; };
8 {Ordering} Orderings = ....;
9 tuple Part { Task task; int id; }
10 {Part} Parts = { <t,i> | t in Tasks, i in 1 .. t.dur div t.smin };
11 tuple Step { int x; int y; }
12 sorted {Step} Steps[t in Tasks] = {<x,0> | x in t.ds} union {<x,1> | x in t.de};
13 stepFunction holes[t in Tasks] = stepwise(s in Steps[t]) {s.y -> s.x; 0};
14 dvar interval tasks[t in Tasks] in 0..500;
15 dvar interval a[p in Parts] optional size p.task.smin..p.task.smax;
16 dvar sequence seq in all(p in Parts) a[p] types all(p in Parts) p.task.loc;

```

- Data reading:
 - Distance matrix

Data



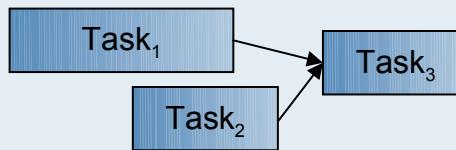
Problem #3: Personal Tasks Scheduling

```

1 using CP;
2 tuple Task { key int id; int loc; int dur; int smin; int smax; int dmin;
3                 int f; int date; {int} ds; {int} de; }
4 {Task} Tasks = ....;
5 tuple Distance { int loc1; int loc2; int dist; };
6 {Distance} Distances = ....;
7 tuple Ordering { int pred; int succ; };
8 {Ordering} Orderings = ....;
9 tuple Part { Task task; int id; }
10 {Part} Parts = { <t,i> | t in Tasks, i in 1 .. t.dur div t.smin };
11 tuple Step { int x; int y; }
12 sorted {Step} Steps[t in Tasks] = {<x,0> | x in t.ds} union {<x,1> | x in t.de};
13 stepFunction holes[t in Tasks] = stepwise(s in Steps[t]) {s.y -> s.x; 0};
14 dvar interval tasks[t in Tasks] in 0..500;
15 dvar interval a[p in Parts] optional size p.task.smin..p.task.smax;
16 dvar sequence seq in all(p in Parts) a[p] types all(p in Parts) p.task.loc;

```

- Data reading:
 - Task ordering



Problem #3: Personal Tasks Scheduling

```

1 using CP;
2 tuple Task { key int id; int loc; int dur; int smin; int smax; int dmin;
3                 int f; int date; {int} ds; {int} de; }
4 {Task} Tasks = ....;
5 tuple Distance { int loc1; int loc2; int dist; };
6 {Distance} Distances = ....;
7 tuple Ordering { int pred; int succ; };
8 {Ordering} Orderings = ....;
9 tuple Part { Task task; int id; }
10 {Part} Parts = { <t,i> | t in Tasks, i in 1 .. t.dur div t.smin };
11 tuple Step { int x; int y; }
12 sorted {Step} Steps[t in Tasks] = {<x,0> | x in t.ds} union {<x,1> | x in t.de};
13 stepFunction holes[t in Tasks] = stepwise(s in Steps[t]) {s.y -> s.x; 0};
14 dvar interval tasks[t in Tasks] in 0..500;
15 dvar interval a[p in Parts] optional size p.task.smin..p.task.smax;
16 dvar sequence seq in all(p in Parts) a[p] types all(p in Parts) p.task.loc;

```

- Data computing:
 - Computation of envelope of possible task parts:
there are at most $t.dur / t.smin$ parts for a task t

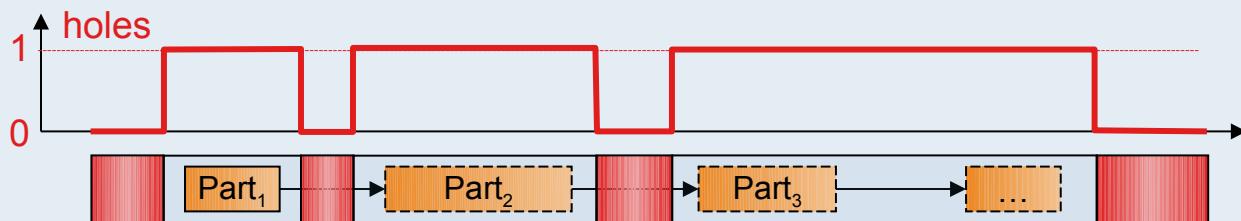
Problem #3: Personal Tasks Scheduling

```

1 using CP;
2 tuple Task { key int id; int loc; int dur; int smin; int smax; int dmin;
3                 int f; int date; {int} ds; {int} de; }
4 {Task} Tasks = ....;
5 tuple Distance { int loc1; int loc2; int dist; };
6 {Distance} Distances = ....;
7 tuple Ordering { int pred; int succ; };
8 {Ordering} Orderings = ....;
9 tuple Part { Task task; int id; }
10 {Part} Parts = { <t,i> | t in Tasks, i in 1 .. t.dur div t.smin };
11 tuple Step { int x; int y; }
12 sorted {Step} Steps[t in Tasks] = {<x,0> | x in t.ds} union {<x,1> | x in t.de};
13 stepFunction holes[t in Tasks] = stepwise(s in Steps[t]) {s.y -> s.x; 0};
14 dvar interval tasks[t in Tasks] in 0..500;
15 dvar interval a[p in Parts] optional size p.task.smin..p.task.smax;
16 dvar sequence seq in all(p in Parts) a[p] types all(p in Parts) p.task.loc;

```

- Data computing:
 - Step functions for task domains



Problem #3: Personal Tasks Scheduling

```

1 using CP;
2 tuple Task { key int id; int loc; int dur; int smin; int smax; int dmin;
3                 int f; int date; {int} ds; {int} de; }
4 {Task} Tasks = ....;
5 tuple Distance { int loc1; int loc2; int dist; };
6 {Distance} Distances = ....;
7 tuple Ordering { int pred; int succ; };
8 {Ordering} Orderings = ....;
9 tuple Part { Task task; int id; }
10 {Part} Parts = { <t,i> | t in Tasks, i in 1 .. t.dur div t.smin };
11 tuple Step { int x; int y; }
12 sorted {Step} Steps[t in Tasks] = {<x,0> | x in t.ds} union {<x,1> | x in t.de};
13 stepFunction holes[t in Tasks] = stepwise(s in Steps[t]) {s.y -> s.x; 0};
14 dvar interval tasks[t in Tasks] in 0..500;
15 dvar interval a[p in Parts] optional size p.task.smin..p.task.smax;
16 dvar sequence seq in all(p in Parts) a[p] types all(p in Parts) p.task.loc;

```

- **Decision variables:**
 - Tasks: array of interval variables
 - Task parts: array of **optional** interval variables
 - **Sequence variable** defined over all task parts

Problem #3: Personal Tasks Scheduling

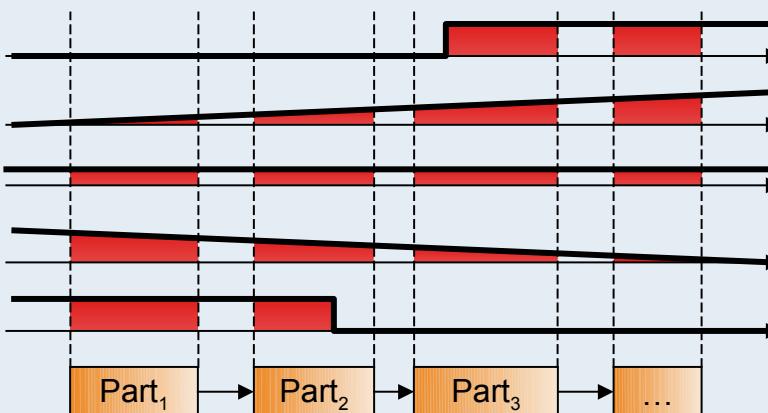
```

17 int L[t in Tasks] = min(x in t.ds) x;
18 int R[t in Tasks] = max(x in t.de) x;
19 int S[t in Tasks] = R[t]-L[t];
20 dexpr float satisfaction[t in Tasks] = (t.f==0) ? 1 :
21   (1/t.dur)* sum(p in Parts: p.task==t)
22   (t.f== 2)? maxl(endOf(a[p]),t.date)-maxl(startOf(a[p]),t.date) :
23   (t.f== 1)? lengthOf(a[p])*((startOf(a[p])+endOf(a[p])-1)/2-L[t])/S[t] :
24   (t.f== -1)? lengthOf(a[p])*(R[t]-(startOf(a[p])+endOf(a[p])-1)/2)/S[t] :
25   (t.f== -2)? minl(endOf(a[p]),t.date)-minl(startOf(a[p]),t.date) : 0;
26 maximize sum(t in Tasks) satisfaction[t];

```

- Objective function:
- Sum of individual task satisfaction
 - 5 types of preference functions (note that some are quadratic)

Objective



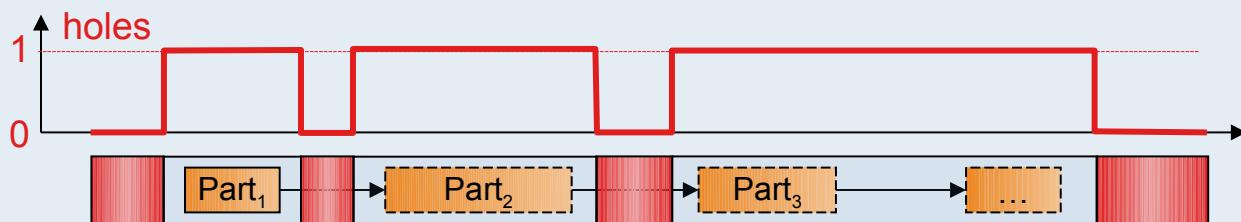
Problem #3: Personal Tasks Scheduling

```

27 subject to {
28   forall(p in Parts) {
29     forbidExtent(a[p], holes[p.task]);
30     forall(s in Parts: s.task==p.task && s.id==p.id+1) {
31       endBeforeStart(a[p], a[s], p.task.dmin);
32       presenceOf(a[s]) => presenceOf(a[p]);
33     }
34   }
35   forall(t in Tasks) {
36     t.dur == sum(p in Parts: p.task==t) lengthOf(a[p]);
37     span(tasks[t], all(p in Parts: p.task==t) a[p]);
38   }
39   forall(o in Orderings)
40     endBeforeStart(tasks[<o.pred>], tasks[<o.succ>]);
41   noOverlap(seq, Distances);
42 }
```

- Constraints:

- Task domain

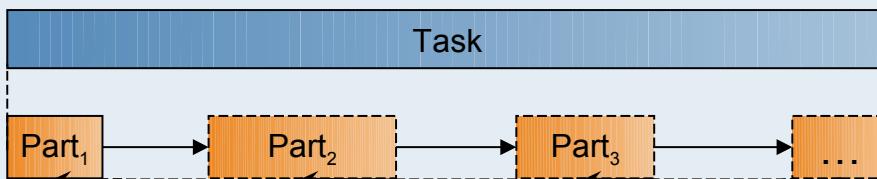


Problem #3: Personal Tasks Scheduling

```

27 subject to {
28   forall(p in Parts) {
29     forbidExtent(a[p], holes[p.task]);
30     forall(s in Parts: s.task==p.task && s.id==p.id+1) {
31       endBeforeStart(a[p], a[s], p.task.dmin);
32       presenceOf(a[s]) => presenceOf(a[p]);
33     }
34   }
35   forall(t in Tasks) {
36     t.dur == sum(p in Parts: p.task==t) lengthOf(a[p]);
37     span(tasks[t], all(p in Parts: p.task==t) a[p]);
38   }
39   forall(o in Orderings)
40     endBeforeStart(tasks[<o.pred>], tasks[<o.succ>]);
41   noOverlap(seq, Distances);
42 }
```

- Constraints:
 - Task decomposition into parts



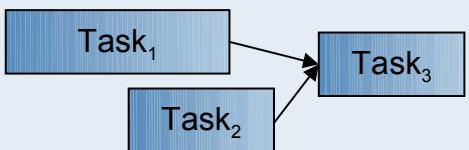
Problem #3: Personal Tasks Scheduling

```

27 subject to {
28   forall(p in Parts) {
29     forbidExtent(a[p], holes[p.task]);
30     forall(s in Parts: s.task==p.task && s.id==p.id+1) {
31       endBeforeStart(a[p], a[s], p.task.dmin);
32       presenceOf(a[s]) => presenceOf(a[p]);
33     }
34   }
35   forall(t in Tasks) {
36     t.dur == sum(p in Parts: p.task==t) lengthOf(a[p]);
37     span(tasks[t], all(p in Parts: p.task==t) a[p]);
38   }
39   forall(o in Orderings)
40     endBeforeStart(tasks[<o.pred>], tasks[<o.succ>]);
41   noOverlap(seq, Distances);
42 }
```

Constraints

- Constraints:
 - Ordering constraints



Problem #3: Personal Tasks Scheduling

```

27 subject to {
28   forall(p in Parts) {
29     forbidExtent(a[p], holes[p.task]);
30     forall(s in Parts: s.task==p.task && s.id==p.id+1) {
31       endBeforeStart(a[p], a[s], p.task.dmin);
32       presenceOf(a[s]) => presenceOf(a[p]);
33     }
34   }
35   forall(t in Tasks) {
36     t.dur == sum(p in Parts: p.task==t) lengthOf(a[p]);
37     span(tasks[t], all(p in Parts: p.task==t) a[p]);
38   }
39   forall(o in Orderings)
40     endBeforeStart(tasks[<o.pred>], tasks[<o.succ>]);
41   noOverlap(seq, Distances);
42 }
```

- Constraints:
 - **No-overlap** constraint between task parts using **transition distance**

Problem #3: Personal Tasks Scheduling

■ Experimental Results

#	SWO	CPO									
15-1	12.95	14.66	30-6	28.09	29.28	40-1	24.72	28.95	45-6	32.70	37.35
15-2	12.25	13.16	30-7	23.80	24.20	40-2	23.48	32.07	45-7	32.40	35.77
15-3	13.71	13.90	30-8	24.06	26.89	40-3	33.57	37.74	45-8	31.79	35.23
15-4	11.57	12.55	30-9	23.42	24.86	40-4	31.46	35.45	45-9	35.79	38.86
15-5	12.64	14.67	30-10	22.04	27.18	40-5	28.05	34.21	45-10	32.78	40.68
15-6	14.30	14.63	35-1	28.80	31.56	40-6	29.46	34.01	50-1	42.04	43.53
15-7	13.08	14.46	35-2	29.17	32.33	40-7	33.13	37.51	50-2	×	×
15-8	11.46	12.37	35-3	27.84	28.58	40-8	29.72	34.90	50-3	×	37.17
15-9	11.44	11.61	35-4	26.64	29.67	40-9	33.03	36.89	50-4	×	36.52
15-10	12.07	13.51	35-5	25.15	32.13	40-10	30.28	34.19	50-5	34.25	43.55
30-1	24.17	29.13	35-6	26.12	29.49	45-1	37.42	42.90	50-6	38.32	41.87
30-2	24.69	27.55	35-7	29.28	31.69	45-2	33.97	39.71	50-7	32.59	42.48
30-3	25.61	26.53	35-8	25.71	30.07	45-3	35.44	39.40	50-8	34.70	43.67
30-4	27.13	28.49	35-9	23.74	29.60	45-4	33.02	37.41	50-9	×	42.75
30-5	23.89	26.46	35-10	30.70	33.41	45-5	30.83	36.65	50-10	37.46	41.84
55-1	×	36.84	55-4	×	40.36	55-7	×	×	55-10	×	×
55-2	×	38.56	55-5	×	42.70	55-8	×	45.27			
55-3	×	×	55-6	×	35.92	55-9	×	42.14			

Table 3. Results for Personal Task Scheduling

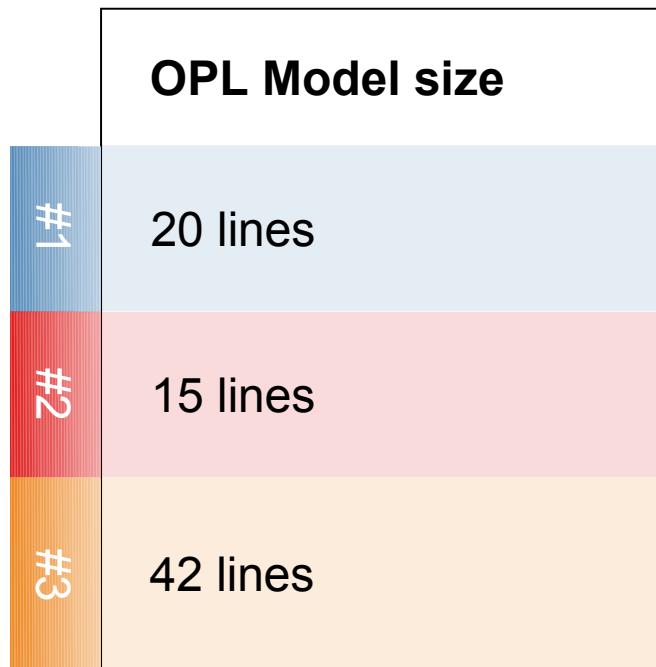
- More solutions found for large instances
- Average task satisfaction increases from 78% to 87% in average (12.5% improvement)

- These problems cover many aspects of scheduling ...

	Domain	Activity type	Resource type	Temporal network	Objective function
#1	Manufacturing	Non-preemptive	Disjunctive	Structured (jobs)	Earliness/tardiness
#2	Aerospace	Non-preemptive	Cumulative	None	Scheduled tasks
#3	Project scheduling	Preemptive	Disjunctive	Unstructured	Complex temporal preferences

- They can easily be modeled with CP Optimizer

...



- They can efficiently be solved by CP Optimizer
- ...

	OPL Model size	CPO Automatic search (no parameter tuning) vs. state-of-the-art
#1	20 lines	Competitive with state of the art (GA, LNS)
#2	15 lines	Number of unscheduled tasks decreased by 5%
#3	42 lines	Finds solution to more instances Solution quality increased by 12.5%

Some more results ... (*compiled in 2007*)

Bench index	Problem type	MRD	# Imp. UBS / # Instances
1	Trolley	-10.2%	15/15
2	Hybrid flow-shop	-11.3%	19/20
3	Job-shop w/ E/T	-6.2%	41/48
4	Air traffic management	-7.0%	1/1
5	Max. quality RCPSP	-2.7%	NA/3600
6	Flow-shop w/ E/T	-1.1%	5/12
7	RCPSP w/ E/T	-2.1%	16/60
8	Cumulative job-shop	-0.1%	15/86
9	Semiconductor testing	-0.3%	7/18
10	Single proc. tardiness	0.3%	0/20
11	Open-shop	0.3%	0/28
12	MaScLib single machine	0.6%	0/60
13	Shop w/ setup times	0.4%	3/15
14	RCPSP	1.2%	2/600
15	Air land	0.0%	0/8
16	Parallel machine w/ E/T	1.6%	4/52
17	Job-shop	1.9%	0/33
18	Flow-shop	0.9%	4/22
19	Flow-shop w/ buffers	3.9%	11/30
20	Single machine w/ E/T	7.4%	0/40
21	Aircraft assembly	8.7%	0/1
22	Common due-date	6.8%	4/20

- Product page:
 - ibm.com/software/integration/optimization/cplex-optimization-studio
 - Trial version
 - Product documentation
 - White papers
 - Presentations
- IBM ILOG Optimization Forums / Constraint Programming:
 - <http://www.ibm.com/developerworks/forums/category.jspa?categoryID=268>

- IBM Academic Initiative
 - **IBM ILOG CPLEX Optimization Studio** is available at no charge for academic use (research and teaching) in the context of the IBM Academic Initiative program
 - Search the web for "[IBM Academic Initiative](#)"

Finding out more ...

- Some papers on CP Optimizer (not limited to scheduling)

P. Laborie, J. Rogerie, P. Shaw and P. Vilim. "**Reasoning with Conditional Time-intervals - Part II: an Algebraical Model for Resources**". In Proceedings of the Twenty-Second International FLAIRS Conference. 2009.

P. Laborie. "**IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems**". In Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'09). 2009.

P. Vilim. "**Max Energy Filtering Algorithm for Discrete Cumulative Resources**". In Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'09). 2009.

P. Vilim. "**Edge finding filtering algorithm for discrete cumulative resources in $O(kn\log(n))$** ". In Proceedings of the 15th international conference on Principles and practice of constraint programming (CP'09). 2009.

P. Laborie and J. Rogerie. "**Reasoning with Conditional Time-intervals**". In Proceedings of the Twenty-First International FLAIRS Conference. pp555-560. 2008.

P. Laborie and D. Godard. "**Self-Adapting Large Neighborhood Search: Application to single-mode scheduling problems**". In Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA). pp276-284. 2007.

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

A. Gargani and P. Refalo. "**An Efficient Model and Strategy for the Steel Mill Slab Design Problem**". In Proceedings of the 13th international conference on Principles and Practice of Constraint Programming (CP'07). 2007.

P. Refalo: "**Impact-Based Search Strategies for Constraint Programming**". In Proceedings of the 10th international conference on Principles and Practice of Constraint Programming (CP'04). 2004.

P. Shaw: "**A Constraint for Bin Packing**". In Proceedings of the 10th international conference on Principles and Practice of Constraint Programming (CP'04). 2004.