Creating Tests for a Family of Cost Aware Resource Constraints

Helmut Simonis and Tarik Hadzic

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

CSCLP 2010, Berlin

science foundation ireland
fondúireacht eolaíochta éireann

Background

- Funded by Science Foundation Ireland (SFI)
- TIDA project: Application oriented research
- Cooperation with IBM, United Technologies
- Objective: Develop scheduling tools for energy cost aware scheduling

Background

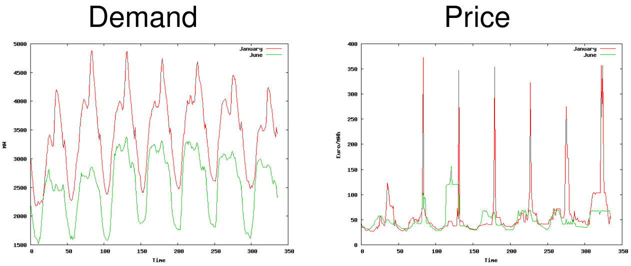


Figure: Electricity Demand and Price, ROI, Comparing January and June data

Background

So far our work has considered the core algorithms to include resource costs into scheduling constraints:

- *An energy cost aware cumulative.* BPPC'10, Bologna, Italy, June 2010.
- *Constraint-based scheduling for reducing peak electricity use.* CompSust'10, Boston, MA, June 2010.
- *A resource cost aware cumulative.* ModRef 2010, St Andrews, Scotland, September 2010.
- *A Family of Resource Constraints for Energy Cost Aware Scheduling*. CROCS 2010, St Andrews, Scotland, September 2010

Background

- We considered an extension of the *Cumulative* constraint
- We extended that work for other resource types: *Disjunctive*, *Parallel-Disjunctive*, *Machine Choice*
- We have proposed different algorithms to compute *lower bounds* on the resource cost of a schedule, and compared their theoretical and practical power
- The best results use an adaptation of the *linear programming model* for the cumulative constraint from [1]. Lower bounds better than 98% of the optimal value can be achieved
- *Reduced cost based filtering* based on these bounds can remove a significant number of domain values

[1] John Hooker, *Integrated Methods for Optimization* (Springer, New York, 2007)

However

- To validate and improve our findings we require test cases on which we can systematically test the performance of our methods
- None of the existing resource constraint benchmarks includes a cost element
- We want to evaluate the quality of different constraint implementations

This Work

Therefore

- We created our own instance generator
- Written in Java
- Produces an XML instance format
- Can be directly integrated into Java code
- Available on the website of the authors

Definition

Constraint `CumulativeCost` expresses the following relationships:

$$\forall\, 0 \leq t < p: \quad pr_t := \sum_{\{i | s_i \leq t < s_i + d_i\}} r_i \leq l \tag{1}$$

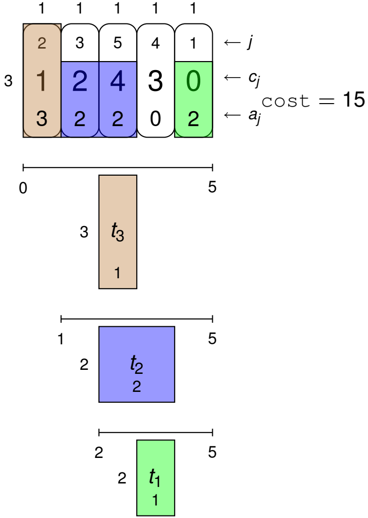
$$\forall\, 1 \leq i \leq n: \quad 0 \leq \underline{s_i} \leq s_i < s_i + d_i \leq \overline{s_i} + d_i \leq p \tag{2}$$

$$\mathrm{ov}(t, pr_t, A_j) := \begin{cases} \max(0, \min(y_j + h_j, pr_t) - y_j) & x_j \leq t < x_j + w_j \\ 0 & \textit{otherwise} \end{cases} \tag{3}$$

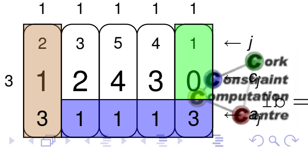
$$\forall\, 1 \leq j \leq q: \quad a_j = \sum_{0 \leq t < p} \mathrm{ov}(t, pr_t, A_j) \tag{4}$$

$$\mathrm{cost} = \sum_{j=1}^{q} a_j c_j \tag{5}$$

Problem and Optimal Solution



LP Solution



LP Lower Bound

$$\texttt{lb} = \min \sum_{j=1}^{q} a_j c_j \quad (6)$$

$$\forall\, 0 \le t < p: \quad pr_t \in [0, l] \quad (7)$$

$$\forall\, 1 \le i \le n, 0 \le t < p: \quad y_{it} \in \{0, 1\} \quad (8)$$

$$\forall\, 1 \le j \le q, \forall\, x_j \le t < x_j + w_j: \quad z_{jt} \in [0, h_j] \quad (9)$$

$$\forall\, 1 \le j \le q: \quad 0 \le \underline{a_j} \le a_j \le \overline{a_j} \le w_j h_j \quad (10)$$

$$\forall\, 1 \le i \le n: \quad s_i = \sum_{t=0}^{p-1} t y_{it} \quad (11)$$

$$\forall\, 1 \le i \le n: \quad \sum_{t=0}^{p-1} y_{it} = 1 \quad (12)$$

$$\forall\, 0 \le t < p: \quad pr_t = \sum_{1 \le i \le n} \sum_{t' \le t < t' + d_i} y_{it'} r_i \quad (13)$$

$$\forall\, 0 \le t < p: \quad pr_t = \sum_{j=1}^{q} z_{jt} \quad (14)$$

$$\forall\, 1 \le j \le q: \quad a_j = \sum_{t=x_j}^{x_j + w_j - 1} z_{jt} \quad (15)$$

Cork Constraint Computation Centre

The `DisjunctiveCost` constraint allows one task to be run on a machine at any time. We can describe the constraint by adding

$$\forall\, i, j | i \neq j : \quad s_i + d_i \leq s_j \vee s_j + d_j \leq s_i \tag{16}$$

to constraints (1) -(5).
In the LP/MIP model, we extend constraints (6) - (15) with the condition

$$\forall\, 0 \leq t < p : \quad \sum_{1 \leq i \leq n} \sum_{t' \leq t < t' + d_i} y_{it'} \leq 1 \tag{17}$$

The `ParallelMachineCost` constraint consists of constraints (1) -(5) plus the constraints

$$\forall\, 1 \leq k \leq b,\ \forall\, i,j | i \neq j : \quad m_i \neq m_j \vee s_i + d_i \leq s_j \vee s_j + d_j \leq s_i \tag{18}$$

We can express this condition in the LP/MIP model by adding constraints of the form

$$\forall\, 1 \leq k \leq d,\ \forall\, 0 \leq t < p : \quad \sum_{\{i | m_i = k\}} \sum_{t' \leq t < t' + d_i} y_{it'} \leq 1 \tag{19}$$

to constraints (6) - (15).

Cork Constraint Computation Centre

Definition

Constraint `MachineChoiceCost` expresses the following relationships:

$$\forall\, 0 \le t < p: \quad pr_t := \sum_{\{i | s_i \le t < s_i + d_{im_i}\}} r_{im_i} \le l \tag{20}$$

$$\forall\, 1 \le i \le n: \quad 0 \le \underline{s_i} \le s_i < s_i + d_{im_i} \le \overline{s_i} + d_{im_i} \le p \tag{21}$$

$$\mathrm{ov}(t, pr_t, A_j) := \begin{cases} \max(0, \min(y_j + h_j, pr_t) - y_j) & x_j \le t < x_j + w_j \\ 0 & \textit{otherwise} \end{cases} \tag{22}$$

$$\forall\, 1 \le j \le q: \quad a_j = \sum_{0 \le t < p} \mathrm{ov}(t, pr_t, A_j) \tag{23}$$

$$\mathtt{cost} = \sum_{j=1}^{q} a_j c_j \tag{24}$$

Instance Generator

The instance generator is available in the form of a Java jar file *CostInstance.jar* which can be downloaded from

`http://4c.ucc.ie/~thadzic/CostInstance.jar`

To create an instance, execute:

```
java -cp CostInstance.jar Instance <parameters>
```

where <*parameters*> are:

Instance Generator

`-instanceType`	0 - `CumulativeCost`, 1 - `DisjunctiveCost`, 2 - `ParallelMachineCost`
`-n`	number of required tasks
`-m`	number of required areas
`-d_max`	maximum duration of a task
`-r_max`	maximum resource consumption of a task
`-s_diff_portion`	portion of the horizon restricting the start time domain
`-util`	utilization of the total available area
`-cost_distr`	cost distribution, 0 - explicitly given, 1 - random
`-w`	width of each area
`-machineNo`	number of machines for parallel machine instances
`-randomSeed`	initial random seed
`-maxCost`	maximal random cost of an area
`-costFileName`	a valid file name containing a vector of costs (or input "`no-file`")

Parameter: `s_diff_portion`

Let p denote the horizon and s_{max} denote $p \cdot$ `s_diff_portion`. Then the instance generator enforces:

$$0 \leq \underline{s_i} \leq s_i \leq \overline{s_i} \leq \underline{s_i} + s_{max}$$

If `s_diff_portion` is set to a negative value then each task i has a maximal feasible start-time interval $[0, p - d_i]$.

Cork Constraint Computation Centre

Parameter: `util`

An integer value between 0 and 100 designating the percentage of the required *utilization*. For `CumulativeCost` and `ParallelMachineCost` the instance generator enforces:

$$l = \frac{\sum_{i=1}^{n} d_i \cdot r_i}{\mathtt{util} \cdot p} \tag{25}$$

where l is a maximal resource level. For `DisjunctiveCost` the generator enforces:

$$p = \frac{\sum_{i=1}^{n} \mathrm{d}_i}{\mathtt{util}}. \tag{26}$$

Note that due to rounding to integer values, exact utilization levels might not be achieved.

Parameter: `cost_distr`

- For *random cost distribution* (`cost_distr`=1), we select a cost for each area from interval $[0, \texttt{maxCost} - 1]$.
- For *explicitly given costs* (`cost_distr`=0), we cyclically repeat vector of costs given in the file `costFileName`. For example, for a vector of costs 10, 20, 30 and $m = 5$ areas, we generate costs $10, 20, 30, 10, 20$.
- For *explicitly given costs* (`cost_distr`=0) and no external file (`costFileName = "no-file"`) we use default, hardcoded vector of electricity costs which consists of 48 values, covering 24 hour period for a particular day on the Irish electricity market.

Cork Constraint Computation Centre

XML Output Example


```
<?xml version = "1.0" encoding = "UTF − 8" standalone = "yes"? >
< instance resource − limit = "6" horizon = "21"
  xmlns : xsi = "http : //www.w3.org/2001/XMLSchema − instance"
  xsi : noNamespaceSchemaLocation = "resourcecost.xsd"/ >
< tasks number = "5" >
< task id = "0" start_min = "4" start_max = "4" duration = "3" resource = "2"/ >
< task id = "1" start_min = "1" start_max = "5" duration = "4" resource = "5"/ >
< task id = "2" start_min = "8" start_max = "8" duration = "6" resource = "5"/ >
< task id = "3" start_min = "0" start_max = "10" duration = "3" resource = "2"/ >
< task id = "4" start_min = "3" start_max = "9" duration = "3" resource = "3"/ >
< /tasks >
< areas number = "7" >
< area id = "0" x = "0" y = "0" width = "3" height = "6" cost = "14"/ >
< area id = "1" x = "3" y = "0" width = "3" height = "6" cost = "11"/ >
< area id = "2" x = "6" y = "0" width = "3" height = "6" cost = "7"/ >
< area id = "3" x = "9" y = "0" width = "3" height = "6" cost = "7"/ >
< area id = "4" x = "12" y = "0" width = "3" height = "6" cost = "0"/ >
< area id = "5" x = "15" y = "0" width = "3" height = "6" cost = "7"/ >
< area id = "6" x = "18" y = "0" width = "3" height = "6" cost = "7"/ >
< /areas >
< machines number = "2" >
< machine id = "0" tasks = "2 4 3"/ >
< machine id = "1" tasks = "0 1"/ >
< /machines >
```

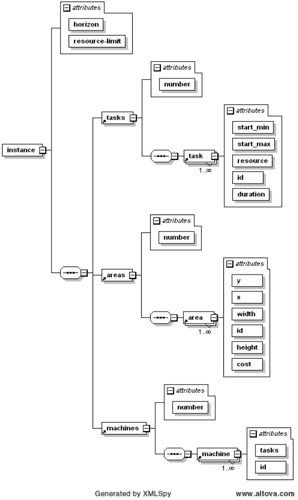


Figure: The XML schema, describing the output format of the output XML file.

Existing Benchmarks for Resource Constraints

- To the best of our knowledge, this is the first generator of scheduling instances involving cost-aware resources
- A number of scheduling instances were collected previously:
  - Patterson: A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem,1984
  - Alvarez et al: Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis, 1989
  - Taillard: Benchmarks for basic scheduling problems,1993
  - Kolisch et al: Characterization and generation of a general class of resource-constrained project scheduling problems, 1995
  - Baptiste and Le Pape: Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems, 2000

Cork Constraint Computation Centre

Existing Benchmarks for Resource Constraints

- The most comprehensive instance generator is presented in the body of work by Kolisch et al.
    - Authors introduce instance generator *ProGen* for the general class of *project scheduling instances*
    - The generator produces *multi-modal* instances over *multiple resources*
    - Tightness of *precedence relationships* is controlled by generating the *activity-on-node* network
    - Scarcity level of resource is controlled through a *resource strength*, $RS \in [0, 1]$.
- In comparison, we could classify our generator as producing *cost-aware*, *single-modal* instances over a *single resource*, *without precedence relationships*. Instead, our generator uses *restricted start times* (`s_diff_portion`) and *utilization* parameter (`util`) to control the tightness and the scarcity level.

Integration into JSR-331

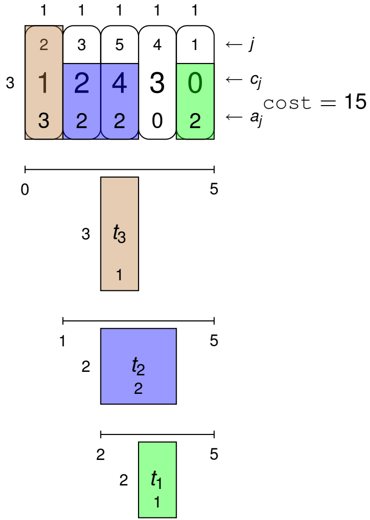
We are integrating our code into the *JSR-331: Constraint Programming API*. We extended a standard standard *Resource* class with *ResourceWithCost* class that supports a cost-declaring method:

```
Var setCost(int x1,int x2,int y1,int y2,int cost)
```

Cork Constraint Computation Centre

Our Working Example in JSR-331

Problem and Optimal Solution



Our Working Example in JSR-331

```
setStart(0);
setEnd(5);

Activity T1 = activity("Task1", 1);
Activity T2 = activity("Task2", 2);
Activity T3 = activity("Task3", 1);
post(T1, ">=", 2);
post(T2, ">=", 1);

ResourceWithCost myResource = new ResourceWithCost(this, 3);
T1.requires(myResource, 2);
T2.requires(myResource, 2);
T3.requires(myResource, 3);

int[] x1 = {0,1,2,3,4};
int[] x2 = {1,2,3,4,5};
int[] y1 = {0,0,0,0,0};
int[] y2 = {3,3,3,3,3};
int[] cost = {1,2,4,3,0};
```

Cork Constraint Computation Centre

Our Working Example in JSR-331

```
utilVars = new Var[5];
for (int i = 0; i < utilVars.length; i++) {
 utilVars[i] = myResource.setCost(x1[i],x2[i],y1[i],y2[i],cost[i]);
}
Var totalCost = scalProd(cost, utilVars);
```

Integration of Instance Generator into JSR-331

```
Instance instance;

public void createInstance() {
 int instanceType = 0;
 int n = 50;
 int m = 10;
 int d_max = 7;
 int r_max = 5;
 double s_diff_portion = 0.5;
 int util = 50;
 int cost_distr = 1;
 int units_per_area = 1;
 int machineNo = 1;
 long randomSeed = 123456;
 int max_cost = 10;
 String costFileName = new String("no-file");

 instance = new Instance(instanceType, n, m, d_max, r_max,
                         s_diff_portion, util, cost_distr,
                         units_per_area, machineNo, randomSeed,
                         max_cost, costFileName);
}
```

Integration of Instance Generator into JSR-331

```
setStart(0);
setEnd(instance.p);

ArrayList<Activity> Tasks = new ArrayList<Activity>();
for (int i=0; i<instance.task.length; i++){
    Activity Ti = activity("Task_"+i, instance.task[i].d);
    post(Ti, ">=", instance.task[i].s_min);
    post(Ti, "<=", instance.task[i].s_max);
    Tasks.add(Ti);
}

ResourceWithCost myResource = new ResourceWithCost(this, instance.l);

for (int i=0; i<instance.task.length; i++){
    Tasks.get(i).requires(myResource, instance.task[i].r);
}
```

Cork Constraint Computation Centre

Integration of Instance Generator into JSR-331

```
utilVars = new Var[instance.area.length];
int[] areaUnitCosts = new int[instance.area.length];
for (int i = 0; i < utilVars.length; i++) {
    int x1 = instance.area[i].x;
    int x2 = instance.area[i].x + instance.area[i].w;
    int y1 = instance.area[i].y;
    int y2 = instance.area[i].y + instance.area[i].h;
    int cost = instance.area[i].c;
    areaUnitCosts[i] = cost;
    utilVars[i] = myResource.setCost(x1, x2, y1, y2, cost);
}

Var totalCost = scalProd(areaUnitCosts, utilVars);
```

Conclusions

- We created an instance generator for cost-aware scheduling problems
- Generator produces instances in XML format
- It can be integrated directly into Java code
- Publicly available at

  `http://4c.ucc.ie/~thadzic/CostInstance.jar`
- Currently being integrated into the JSR-331 Constraint Programming API

Cork Constraint Computation Centre