

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



- 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



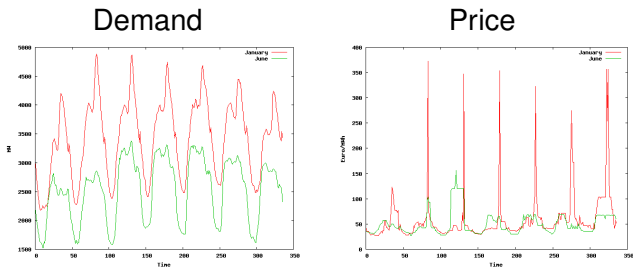


Figure: Electricity Demand and Price, ROI, Comparing January and June data

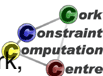
So far our work has considered the core algorithms to include resource costs into scheduling constraints:

- *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



- 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



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:

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

$$\forall 1 \leq i \leq n: \quad 0 \leq \underline{s}_i \leq s_i < s_i + d_i \leq \bar{s}_i + d_i \leq p \quad (2)$$

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

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

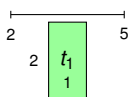
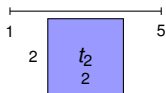
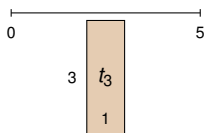
$$\text{cost} = \sum_{j=1}^q a_j c_j \quad (5)$$

An Example

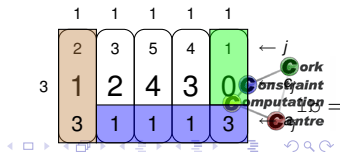
Problem and Optimal Solution

	1	1	1	1	1	
	2	3	5	4	1	$\leftarrow j$
3	1	2	4	3	0	$\leftarrow c_j$
	3	2	2	0	2	$\leftarrow a_j$

cost = 15



LP Solution



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

$$\forall 0 \leq t < p: pr_t \in [0, 1] \quad (7)$$

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

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

$$\forall 1 \leq j \leq q: 0 \leq \underline{a}_j \leq a_j \leq \bar{a}_j \leq w_j h_j \quad (10)$$

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

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

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

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

$$\forall 1 \leq j \leq q: a_j = \sum_{t=x_j}^{x_j+w_j-1} z_{jt}$$

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

$$\forall i, j | i \neq j: \quad s_i + d_i \leq s_j \vee s_j + d_j \leq s_i \quad (16)$$

to constraints (1) -(5).

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

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



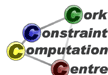
The `ParallelMachineCost` constraint consists of constraints (1) - (5) plus the constraints

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

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

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

to constraints (6) - (15).



Definition

Constraint `MachineChoiceCost` expresses the following relationships:

$$\forall 0 \leq t < p: \quad pr_t := \sum_{\{i | s_i \leq t < s_i + d_{im_i}\}} r_{im_i} \leq l \quad (20)$$

$$\forall 1 \leq i \leq n: \quad 0 \leq \underline{s}_i \leq s_i < s_i + d_{im_i} \leq \bar{s}_i + d_{im_i} \leq p \quad (21)$$

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

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

$$\text{cost} = \sum_{j=1}^q a_j c_j \quad (24)$$

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:

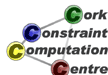
```
java -cp CostInstance.jar Instance <parameters>
```

where <parameters> are:



Instance Generator

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



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 $[0, p - d_i]$.



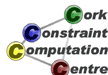
An integer value between 0 and 100 designating the percentage of the required *utilization*. For `CumulativeCost` and `ParallelMachineCost` the instance generator enforces:

$$I = \frac{\sum_{i=1}^n d_i \cdot r_i}{\text{util} \cdot p} \quad (25)$$

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

$$p = \frac{\sum_{i=1}^n d_i}{\text{util}}. \quad (26)$$

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

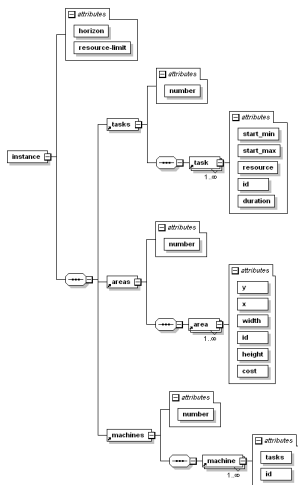


- For *random cost distribution* (`cost_distr=1`), we select a cost for each area from interval $[0, \text{maxCost} - 1]$.
- For *explicitly given costs* (`cost_distr=0`), we cyclically repeat vector of costs given in the file `costFileName`. For example, for a vector of costs 10, 20, 30 and $m = 5$ 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.

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 = "3" 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 >
```

XML Schema



Generated by XMLSpy

www.altova.com

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

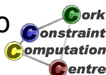
Existing Benchmarks for Resource Constraints

- 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



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.



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:

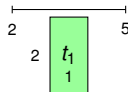
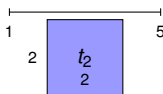
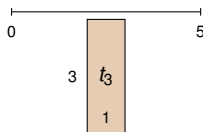
```
Var setCost(int x1,int x2,int y1,int y2,int cost)
```



Problem and Optimal Solution

	1	1	1	1	1	
	2	3	5	4	1	$\leftarrow j$
3	1	2	4	3	0	$\leftarrow c_j$
	3	2	2	0	2	$\leftarrow a_j$

cost = 15



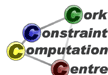
Our Working Example in JSR-331

```
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};
```



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);  
}
```

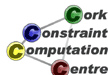


Integration of Instance Generator into JSR-331

```
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);  
}
```

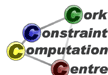
```
ResourceWithCost myResource = new ResourceWithCost(this, instance.l);  
  
for (int i=0; i<instance.task.length; i++){  
    Tasks.get(i).requires(myResource, instance.task[i].r);  
}
```



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);
```



- 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

