Creating Tests for a Family of Cost Aware Resource Constraints

Tarik Hadzic and Helmut Simonis*

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{t.hadzic,h.simonis}@4c.ucc.ie

Abstract. In this paper we describe a benchmark generator for a family of cost aware resource constraints. These constraints have been introduced in the context of electricity cost minimization, but can also be applied for other use cases like manpower scheduling. None of the existing resource constraint benchmarks include a cost element, we therefore created our own test generator to evaluate the quality of different constraint implementations. The generator is written in Java, produces an XML instance format, and is available on the website of the authors.

1 Motivation

Time variable electricity tariffs have been introduced in many countries to properly price the cost of electricity generation over changing demand at different times of the day. In Ireland, the All Island Electricity Market (`http://allislandmarket.com`) generates a whole-sale market price in half hour time slots, with prices sometimes varying by a factor of ten between peak and non-peak periods. The high cost at peak utilization is linked to the use of inefficient, expensive stand-by generation plant with an increased carbon footprint. Shifting large-scale customer demand away from peak utilization increases stability of the national grid, and can possibly delay or avoid heavy investment in new generator capacity.

Figure 1 shows demand and price data for two one week periods in January and June 2010. We can see that demand varies significantly during the day, but also from day to day in the week and between seasons. Prices follow the demand, with especially high prices at periods close to the national dispatch limit.

In Ireland, wind power plays an increasing role as a renewable energy source. Unfortunately, its integration in the national grid is problematic, due to the relative isolation of the Irish electricity grid, and wind energy's time variable, and only partially known, supply level. Matching electricity use to changes in wind energy supply is quite difficult to imagine for domestic usage, but can be possible for industrial users, given the right pricing incentive. Widespread

* This work was supported by Science Foundation Ireland (Grant Numbers 05/IN/I886 and 05/IN.1/I886 TIDA 09).

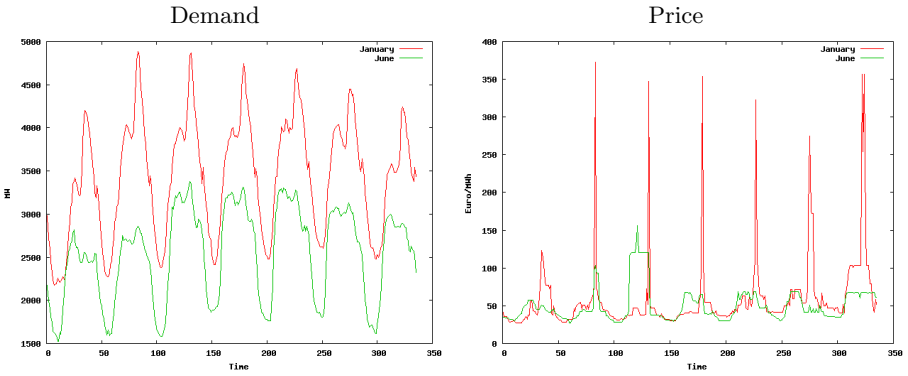


Fig. 1. Electricity Demand and Price, ROI, Comparing January and June data

adaption of electric cars is seen as another way of fully utilizing the wind energy potential, but this is still years in the future.

The current adaption of time variable tariffs for large scale electricity users is still quite limited. This is partially due to the cost of the required smart metering tools and the necessary detailed understanding of electrical usage over short time periods, but also due to a lack of decision support tools which allow the exploitation of time variable tariffs to reduce costs. We are trying to fill this gap using constraint programming.

Our current work is therefore focused on the problem of solving industrial scheduling problems considering time variable energy cost, and using available renewable energy in the most efficient way. This requires agile scheduling tools, which can react quickly to changes in prices, while still considering other optimization criteria like product quality, factory throughput and just-in-time production.

Constraint Programming has been a very successful tool in solving large scale industrial scheduling problems [3,13,14], creating flexible scheduling tools for industries in many domains. Its main advantage over competing technologies is the ease of adapting a system to a changing environment, and the potential to incorporate user-defined strategies and heuristics besides powerful mathematical reasoning techniques.

So far our work [11,12,10] has considered the core algorithms to include resource costs into scheduling constraints. We have initially considered an extension of the cumulative constraint, and then extended that work for other resource types, disjunctive machines, multiple parallel disjunctive machines and machine-choice resource types. We have proposed different algorithms to compute lower bounds on the reosurce cost of a schedule, and compared their theorectical and practical power. The best results use an adaptation of the Linear Programming model for the cumulative constraint from [5]. Experimental results indicate that very good estimates (better than 98% of the optimal value) can be achieved. We

have used that lower bound on the cost to perform reduced cost based filtering for the different resource types described. Initial results showed that a significant number of domain values can be removed by this filtering. To validate and improve these fiding we require test cases on which we can systematically test the performance of our methods. As none of the existing scheduling benchmarks consider resource cost, we decided to create our own instance generator and make it publically available. This paper defines the problem domain, introduces the instance generator and describes its capabilities, and compares it to the existing benchmark tests for constraint-based scheduling.

Our current methods are focused on industrial scheduling problems for large scale electricity consumers in manufacturing and other industries. But the underlying technology is also applicable in other areas of significant electricity usage like HVAC (heating/ventilation/air-conditioning) for buildings, or cooling systems in the food industry or for data centres. These are domains where there also is significant overlap with other projects at our institute (4C), and existing collaboration within UCC and with industrial partners. Note that these problem types are quite different from the project scheduling problems that are considered in many of the existing scheduling benchmarks.

2 Constraints Family

We consider four variants of the energy cost aware scheduling constraints:

CumulativeCost The total energy consumption is limited by a hard limit, and each task consumes a fixed amount of energy during its execution. The cost is the total cost of energy consumed over time, priced at different levels.

DisjunctiveCost This constraint models a disjunctive machine, where tasks consume different amounts of energy. The order of the tasks on the machine will therefore affect total energy cost.

ParallelMachineCost This considers multiple disjunctive machines, which all contribute to an overall energy limit. Tasks are fixed on their machine, only the order of the tasks can be affected.

MachineChoiceCost We consider multiple disjunctive machines with an overall energy use limit. Tasks can move between machines, with possibly different duration and resource use on every machine.

We now discuss each of the possible constraints in more detail.

2.1 CumulativeCost

We start with the CumulativeCost constraint, which looks at the overall energy consumption of a set of tasks over time. It is an extension of the classical cumulative constraint [1]

$$\texttt{Cumulative}([s_1, s_2, ...s_n], [d_1, d_2, ...d_n], [r_1, r_2, ...r_n], l, p),$$

describing n tasks with start s_i, fixed duration d_i and resource use r_i, with an overall resource limit l and a scheduling period end p. Our new constraint is denoted as

$$\texttt{CumulativeCost}(\texttt{Areas}, \texttt{Tasks}, l, p, \texttt{cost}).$$

The *Areas* argument is a collection of q areas $\{A_1, \ldots, A_q\}$, which do not overlap and partition the entire available resource area $[0, p] \times [0, l]$. Each area A_j has a fixed position x_j, y_j, fixed width w_j and height h_j, and fixed per-unit cost c_j. Consider the example in Fig. 2 (left). It has 5 areas, each of width 1 and height 3. Our definition allows that an area A_j could start above the bottom level ($y_j > 0$). This reflects the fact that the unit-cost does not only depend on the time of resource consumption but also on its volume. In our electricity example, in some environments, a limited amount of renewable energy may be available at low marginal cost, generated by wind-power or reclaimed process heat. Depending on the tariff, the electricity price may also be linked to the current consumption, enforcing higher values if an agreed limit is exceeded.

We choose the numbering of the areas so that they are ordered by non-decreasing cost ($i \leq j \Rightarrow c_i \leq c_j$); in our example costs are $0, 1, 2, 3, 4$. There could be more than one area defined over the same time slot t (possibly spanning over other time-slots as well). If that is the case, we require that the area "above" has a higher cost. The electricity consumed over a certain volume threshold might cost more.

The *Tasks* argument is a collection of n tasks. Each task T_i is described by its start s_i (between its earliest start $\underline{s_i}$ and latest start $\overline{s_i}$), and fixed duration d_i and resource use r_i. In our example, we have three tasks with durations $1, 2, 1$ and resource use $2, 2, 3$. The initial start times are $s_1 \in [2, 5]$, $s_2 \in [1, 5]$, $s_3 \in [0, 5]$. For a given task allocation, variable a_j states how many resource units of area A_j are used. For the optimal solution in our example we have $a_1 = 2, a_2 = 3, a_3 = 2, a_4 = 0, a_5 = 2$. Finally, we can define our constraint:

Definition 1. *Constraint* `CumulativeCost` *expresses the following relationships:*

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

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

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

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

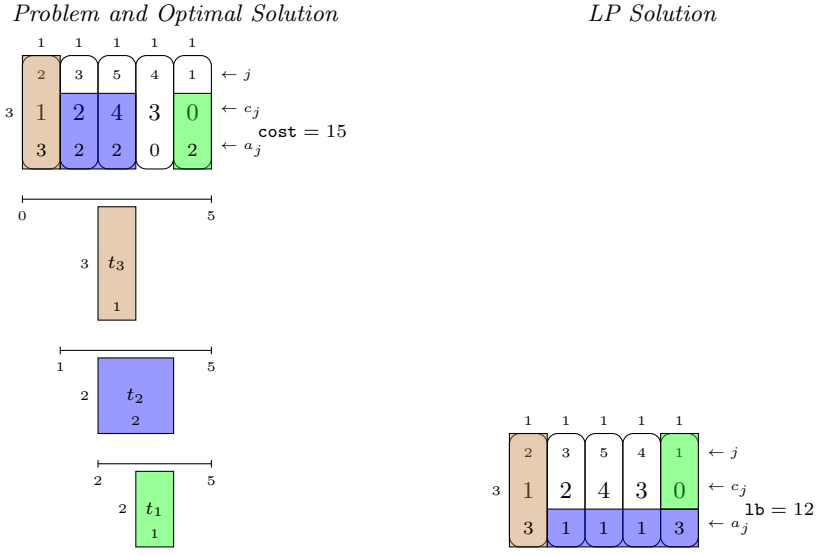
$$\texttt{cost} = \sum_{j=1}^{q} a_j c_j \tag{5}$$

For each time point t we first define the resource profile pr_t (the amount of resource consumed at time t). That profile must be below the overall resource limit

l, as in the standard cumulative. The term $\texttt{ov}(t, pr_t, A_j)$ denotes the intersection of the profile at time t with area A_j, and the sum of all such intersections is the total resource usage a_j. The cost is computed by weighting each intersection a_j with the per-unit cost c_j of the area.

Note that our constraint is a strict generalization of the standard cumulative. Since enforcing generalized arc consistency (GAC) for `Cumulative` is NP-hard [5], the complexity of enforcing GAC over `CumulativeCost` is NP-hard as well.

Fig. 2. An example with 3 tasks and 5 areas. Areas are drawn as rounded rectangles at the top, the tasks to be placed below, each with a line indicating its earliest start and latest end. The optimal placement of tasks has cost 15 (left). The LP model produces a lower bound 12 (right).



In [12] we considered different algorithms to compute a lower bound on the resource cost of the CumulativeCost constraint. The best model was based on [5], which describes an LP relaxation of the classical cumulative constraint. We extend this model to handle the cost component directly (equations (6)-(15)).

We introduce binary variables y_{it} which state whether task i starts at time t. For each task, exactly one of these variables will be one (constraint (12)). Equations (11) connect the s_i and y_{it} variables. Continuous variables pr_t describe the resource profile at each time point t, all values must be below the resource limit l. The profile is used in two ways: In (13), the profile is built by cumulating all active tasks at each time-point. In (14), the profile overlaps all areas active at a time-point, where the contribution of area j at time-point t is called z_{jt} (a

continuous variable ranging between zero and h_j). Adding all contributions of an area leads to the resource use a_j for area j. This model combines the start-time based model of the cumulative with a standard LP formulation of the convex, piece-wise linear cost of the resource profile at each time point. Note that this model relies on the objective function to fill up cheaper areas to capacity before using more expensive ones. Enforcing the integrality in (8) leads to a mixed integer programming model *DMIP*, relaxing the integrality constraint leads to the LP model *DLP*. The MIP model solves the cumulative-cost constraint to optimality, thus providing an exact bound for the constraint. We can ignore the actual solution if we want to use the constraint in a larger constraint problem.

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

$$\forall\ 0 \le t < p: \quad pr_t \in [0, l] \tag{7}$$

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

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

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

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

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

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

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

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

2.2 DisjunctiveCost

The `DisjunctiveCost` constraint is the analog generalization of the disjunctive constraint, which allows one task run be run on a machine at any time. Given the areas and tasks as defined before, we can describe the constraint by adding

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

to constraints (1) -(5).

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

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

which states that at each time point only one task can be active. Note that the overall resource limit l becomes meaningless, as one only task consumes resources at any time point. The overall resource limit can be checked a priori by comparing it against the resource requirements r_i of the tasks. There is still a minimization problem arranging the tasks in a sequence such that total energy cost is minimal.

This constraint is useful to model a factory which contains a single, disjunctive resource as the main energy consumer. Modelling that machine as a CumulativeCost resource (together with a finite domain disjunctive constraint, say) will lead to a massive underestimation of the energy cost required.

2.3 ParallelMachineCost

In the previous section we considered a situation where a single disjunctive machine dominates the energy consumption. We now consider a case where b disjunctive machines run in parallel, and each task is fixed to one of those disjunctive machines. Let $1 \leq m_i \leq b$ be the machine on which task i is assigned. The `ParallelMachineCost` constraint then 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).

This `ParallelMachineCost` describes how all tasks compete for energy, and tasks on a single machine must be scheduled with overlap. This model produces stronger bounds than a `CumulativeCost` constraint on the tasks alone, as it considers the disjunctive behaviour as well.

2.4 MachineChoiceCost

The final variant of the constraint family we consider looks at a situation where we have b machines, and tasks can be assigned on alternative machines, possibly with a different duration and resource consumption on each machine. Let d_{ik} be the duration of task i on machine k, and r_{ik} the energy demand for task i on machine k. The variable $m_i \in M_i$ denotes the machine on which task i has been assigned. It must take a value in the set of possible machines M_i for task i.

Definition 2. *Constraint* `MachineChoiceCost` *expresses the following relationships:*

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

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

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

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

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

We use 0/1 variables y_{itk} to state that task i starts at time t on machine k. If a machine can not run on some machine k, then all entries y_{itk} for that machine will be zero.

$$\mathtt{lb} = \min \sum_{j=1}^{q} a_j c_j \tag{25}$$

$$\forall\ 0 \le t < p: \quad pr_t \in [0, l] \tag{26}$$

$$\forall\ 1 \le k \le b, \forall\ 1 \le i \le n, 0 \le t < p: \quad y_{itk} \in \{0, 1\} \tag{27}$$

$$\forall\ 1 \le j \le q, \forall\ x_j \le t < x_j + w_j: \quad z_{jt} \in [0, h_j] \tag{28}$$

$$\forall\ 1 \le j \le q: \quad 0 \le \underline{a_j} \le a_j \le \overline{a_j} \le w_j h_j \tag{29}$$

$$\forall\ 1 \le i \le n: \quad s_i = \sum_{1 \le k \le d} \sum_{t=0}^{p-1} t y_{itk} \tag{30}$$

$$\forall\ 1 \le i \le n: \quad \sum_{1 \le k \le b} \sum_{t=0}^{p-1} y_{itk} = 1 \tag{31}$$

$$\forall\ 0 \le t < p: \quad pr_t = \sum_{1 \le k \le b} \sum_{t' \le t < t' + d_{ik}} y_{it'k} r_{ik} \tag{32}$$

$$\forall\ 0 \le t < p: \quad pr_t = \sum_{j=1}^{q} z_{jt} \tag{33}$$

$$\forall\ 1 \le j \le q: \quad a_j = \sum_{t=x_j}^{x_j + w_j - 1} z_{jt} \tag{34}$$

$$\forall\ 1 \le k \le b,\ \forall\ 0 \le t < p: \quad \sum_{1 \le i \le n} \sum_{t' \le t < t' + d_{ik}} y_{it'k} \le 1 \tag{35}$$

3 An Instance Generator

This section explains how to use the instance generator for the cost-scheduling instances used in the ongoing research on energy-efficient scheduling [11,12,10]. 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`.

3.1 Parameters

Once the file `CostInstance.jar` is downloaded, it can be used to generate a scheduling instance by executing:

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

where <*parameters*> are:

`-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`")

We will now discuss each parameter in details.

`instanceType` An integer from the set $\{0, 1, 2\}$ designating the type of the instance. Value 0 represents an instance of `CumulativeCost`, 1 - `DisjunctiveCost` and 2 - `ParallelMachineCost`. At the moment, generation of `MachineChoiceCost` instances is not supported.

`n` An integer denoting the number of required tasks. The resulting instance is guaranteed to have the n tasks.

`m` An integer denoting the number of required areas. It is *not guaranteed* that the resulting instance would have m areas if that would conflict with the required utilization level (see parameter `util`). The number of areas is tightly related to the *horizon* p through parameter `w` (the width of each area): $p = m \cdot w$.

`d_max` An integer denoting the maximal duration of a task. For each task i, duration d_i is randomly selected from $[1, \texttt{d\_max}]$.

`r_max` An integer denoting the maximal resource use of a task. For each task i, resource use r_i is randomly selected from $[1, \texttt{r\_max}]$.

`s_diff_portion` A fractional value denoting the portion of the horizon that restricts the maximal length of the start-time interval $[\underline{s_i}, \overline{s_i}]$ for each task i. Let s_{max} denote $p \cdot \texttt{s\_diff\_portion}$. Then it holds: $0 \leq \underline{s_i} \leq s_i \leq \overline{s_i} \leq \underline{s_i} + s_{max}$ If `s_diff_portion` is set to a negative value than each task i has a maximal feasible start-time interval $[0, p - d_i]$.

`util` An integer value between 0 and 100 designating the percentage of the required *utilization* i.e. the tightness of the instance. The higher the utilization level the harder it is to find a feasible schedule and the more expensive is the cheapest schedule. In case of `CumulativeCost` and `ParallelMachineCost` instances, utilization denotes the required ratio of the total task volume and the total available area

$$\texttt{util} = \frac{\sum_{i=1}^{n} d_i \cdot r_i}{l \cdot p}. \tag{36}$$

For `DisjunctiveCost`, the utilization denotes the required ratio between the total task length and the horizon:

$$\texttt{util} = \frac{\sum_{i=1}^{n} \mathrm{d}_i}{p}. \tag{37}$$

The required level of utilization is achieved by adjusting the overall resource limit l in case of `CumulativeCost` and `ParallelMachineCost`:

$$l = \frac{\sum_{i=1}^{n} d_i \cdot r_i}{\texttt{util} \cdot p} \tag{38}$$

and adjusting the horizon p (i.e. the number of areas $m = \frac{p}{\texttt{w}}$) in case of `DisjunctiveCost`:

$$p = \frac{\sum_{i=1}^{n} \mathrm{d}_i}{\texttt{util}}. \tag{39}$$

As a result, even though the required number of areas can be specified as the input, the actual number of areas in the instance depends on the required utilization level. Please note that since resource limit l and number of areas m have to be integers, due to rounding in equation (38) and (39) the actual utilization can somewhat defer from the desired utilization level.

`cost_distr` An integer from the set $\{0, 1\}$ indicating explicitly given cost distribution for value 0, and random cost distribution for value 1. For random cost distribution, we select a cost for each area from interval $[0, \texttt{maxCost} - 1]$. For generating instances with explicitly given costs, we use externally specified vector of costs that we cyclically repeat (see parameter `costFileName` below). For a given number of areas m and width w we cyclically repeat the vector of costs until the m areas are generated. For example, for a vector of costs 10, 20, 30 and $m = 5$ areas, we generate costs $10, 20, 30, 10, 20$. In case

the external file is not provided (file name input "`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. For both distributions we subtract from the cost of each area the cost of the cheapest area. This ensures that the costs are normalized, i.e. that the cheapest area has a cost 0.

`w` An integer denoting the width of each area. It links the horizon p to the number of areas m through equation $p = m \cdot w$.

`machineNo` An integer denoting the number of machines for `ParallelMachineCost` instances.

`randomSeed` An integer denoting the random seed used for generating tasks and areas (if negative then the current system time is used).

`maxCost` A positive integer indicating a maximal cost of an area in a random cost distribution. The cost of each area is a random number from interval $[0, \texttt{maxCost} - 1]$. If a non-positive value is provided, $\texttt{maxCost} \leq 0$, then default maximal cost is used, $\texttt{maxCost} = 150$.

`costFileName` A file name in which the cost vector is specified. The cost information is used if the parameter `cost_distr` is set to 0. The file should contain a single line with space seperated integers. The integers are then cyclically repeated as costs of m areas. For example, if the file contains three entries: `10 20 30` and we want to generate 5 areas, the respective costs will be: 10, 20, 30, 10, 20. In case the external file is not provided, the input "`no-file`" is expected. In that case, a default, hardcoded vector is used, consisting of 48 values of electricity costs, covering 24 hour period for a particular day on the Irish electricity market.

Discussion Please note that the current version generates only the "non-stacking" areas, i.e. over each time point there is only one area defined. Furthermore, each area has the height equal to the maximal resource availability l. Furthermore, for `DisjunctiveCost` instances, the resource limit l does not influence the utilization, and is set to be higher then the highest resource consumption among the tasks, $l = \max\{r_i + 1 \mid i = 1, \ldots, n\}$. In case of `CumulativeCost` and `ParallelMachineCost` instances, if the size of task volume is small in comparison of the length of horizon, in order to achieve the required utilization, the computed overall resource limit l might be smaller that the resource use of individual tasks, leading to an infeasible instance.

3.2 XML Output Format

The output of the generator is an instance in the XML format. The root of the file is an element `<instance>` with attributes `horizon` (denoting the task period p) and `resource-limit` (denoting the overall resource limit l). The `<instance>` consists of `<tasks>`, `<areas>` and `<machines>`. Element `<tasks>` consist of multiple `<task>` elements, each of which defines a task, through attributes `start_min`

(earliest start time $\underline{s_i}$), `start_max` (latest start time $\overline{s_i}$), `duration` and `resource`. Element `<areas>` consist of multiple `<area>` elements, each of which defines an area through attributes `x` and `y` (defining the coordinates of the lower-left corner), `width`, `height` and `cost`. Finally, element `<machines>` consists of multiple `<machine>` elements, each of which lists the set of tasks that are scheduled to be executed on that machine through attribute `tasks`.

Furthermore, `<tasks>`, `<areas>` and `<machines>` have auxiliary attribute `number` that indicates how many `<task>`, `<area>` and `<machine>` elements are to follow. And finally, each `<task>`, `<area>` and `<machine>` element is uniquely identified through an attribute `id`. In fact, the list of tasks in each machine element is given by listing the corresponding task ids.

The formal specification of the XML output format is given in the XML schema file, whose structure is given in Fig. 3. The file can be downloaded from `http://4c.ucc.ie/~thadzic/resourcecost.xsd`.

As an illustration, consider a result of executing:

```
java -cp CostInstance.jar Instance 2 5 7 6 8 0.5 50 0 3 2 123 150 no-file
```

the generator produces the XML file in Figure 4. The result is an instance of a parallel machine scheduling problem, with overall resource limit $l = 6$ and horizon $p = 21$. The instance has $n = 5$ tasks, with $m = 7$ cost areas, each of width $\mathtt{w} = 3$. The tasks are distributed over $\mathtt{machineNo} = 2$ machines. For each task we specify its start time interval, duration and resource consumption. For each area we specify its left-most coordinate (x, y), its width, its height and its cost per resource unit. For each machine we list the indexes of tasks that are scheduled for execution on such machine.

4 Existing Benchmarks for Resource Constraints

To the best of our knowledge, this is the first generator for cost-aware scheduling instances. However, there are several sources of classical scheduling instances, where typically the objective is the *makespan minimization*.

In [15] the authors describe the set of widely cited flow-shop, job-shop and open-shop scheduling instances, for which the generating code is made available at the online collection of OR data sets, the *OR-Library*.[1] Patterson [9] introduced a set of 110 multi-resource scheduling instances involving between 7 and 50 activities, which presented the accumulation of readily-available instances in the literature at that time. Alvarez et al. [2] introduced 144 instances containing 27, 51 and 103 activities respectively. Kolisch et al. [8] introduced a number of general resource-constrained project scheduling instances using the *ProGen* instance generator. They subsequently extended the instance set into a PSPLIB library that is available online [7]. Subsequently, Baptiste and Le Pape [4] introduced the set of more cumulative instances (the *BL* benchmark) since in most of the existing benchmarks the instances are highly disjunctive (i.e. many pairs of activities cannot execute in parallel).

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

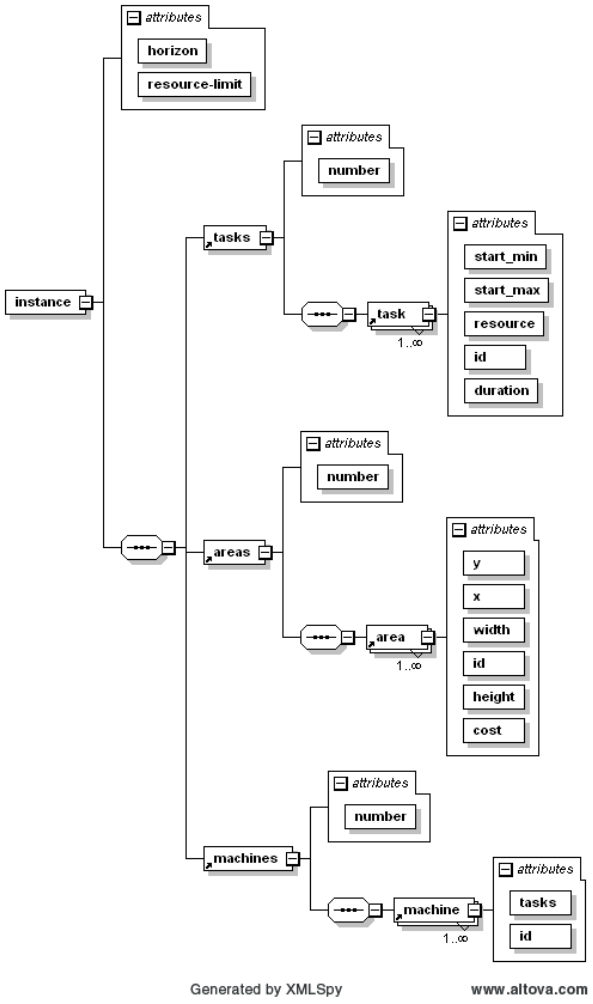


Fig. 3. The XML schema, describing the output format of the output XML file.

```
<?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 >
```

Fig. 4. An xml file output of executing `java -cp CostInstance.jar Instance 2 5 7 6 8 0.5 50 0 3 2 123 150 no-file`

The most comprehensive instance generator is presented in the body of work by Kolisch et al [8,7]. The authors present an instance generator *ProGen* for the general class of *project scheduling instances*. The generator is able to produce *multi-modal* or single-mode instances over multiple resources, where in each mode, each task can consume a different amount or each resource. The instance start times are not controlled by limiting the length of the start time interval. Instead, the tightness of *precedence relationships* is controlled by generating the *activity-on-node* network. The scarcity level of resource is controlled through a *resource strength* (RS) parameter, where $RS \in [0, 1]$. Their method restricted to single-resource instances corresponds roughly to computing the minimal feasible resource consumption $l_{min} = min_{i=1}^{n}\{r_i\}$, and maximal resource consumption l_{max} that corresponds to the peak resource usage for the earliest start schedule. Then, the actual resource limit is generated as a convex combination: $l = l_{min} + RS \cdot (l_{max} - l_{min})$. A good overview of the benchmark library and related datasets is presented in [6].

5 Summary and Future Work

In this paper we presented a Java-based benchmark generator for a family of cost aware resource constraints. To the best of our knowledge, this is the first instance generator for the cost-aware scheduling instances. The generator is written in Java, produces an XML instance format, and is available on the website of the authors.

In future, we plan to extend the generator to support the generation of `MachineChoiceCost` instances as well as other classes of resource-constrained scheduling problems that would be used in the ongoing research.

References

1. A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling problems. *Journal of Mathematical and Computer Modelling*, 17(7):57–73, 1993.
2. Ramon Alvarez-Valdes and Jose Manuel Tamarit. Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis. In R. Slowinski and J. Weglarz, editors, *Advances in project scheduling*, pages 113–134. Elsevier, 1989.
3. P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer, Dortrecht, 2001.
4. Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. In Gert Smolka, editor, *Principles and Practice of Constraint Programming-CP97*, volume 1330 of *Lecture Notes in Computer Science*, pages 375–389. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0017454.
5. John Hooker. *Integrated Methods for Optimization*. Springer, New York, 2007.
6. Rainer Kolisch, Christoph Schwindt, and Arno Sprecher. Benchmark instances for project scheduling problems. In *Handbook on Recent Advances in Project Scheduling*, pages 197–212. Kluwer, 1998.

7. Rainer Kolisch and Arno Sprecher. PSPLIB - a project scheduling problem library. *European Journal of Operational Research*, 96(1):205 – 216, 1997. http://www.sciencedirect.com/science/article/B6VCT-3T7HK9P-1F/2/f3cc7f46a925673bfab16f6be5a4de4b.
8. Rainer Kolisch, Arno Sprecher, and Andreas Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Manage. Sci.*, 41(10):1693–1703, 1995.
9. James H. Patterson. A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem. *MANAGEMENT SCIENCE*, 30(7):854–867, 1984.
10. H. Simonis and T. Hadzic. Constraint-based scheduling for reducing peak electricity use. In *CompSust'10: 2nd International Conference on Computational Sustainability*, Boston, MA, June 2010. http://4c.ucc.ie/~hsimonis/sustain.pdf.
11. H. Simonis and T. Hadzic. An energy cost aware cumulative. In *Third International Workshop on Bin Packing and Placement Constraints BPPC'10*, Bologna, Italy, June 2010. http://4c.ucc.ie/~hsimonis/tidacpaiorabstract.pdf.
12. H. Simonis and T. Hadzic. A resource cost aware cumulative. In *The 9th International Workshop on Constraint Modelling and Reformulation (ModRef 2010)*, September 2010. http://4c.ucc.ie/~hsimonis/tida1.pdf.
13. Helmut Simonis. Building industrial applications with constraint programming. In Hubert Comon, Claude Marché, and Ralf Treinen, editors, *CCL*, volume 2002 of *Lecture Notes in Computer Science*, pages 271–309. Springer, 1999.
14. Helmut Simonis. Models for global constraint applications. *Constraints*, 12(1):63–92, 2007.
15. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278 – 285, 1993. Project Management anf Scheduling.