Modelling a Maintenance Scheduling Problem with Alternative Resources

Aliaa M. Badr[1], Arnaud Malapert[2], and Kenneth N. Brown[1]

[1] Cork Constraint Computation Centre, University College Cork, Ireland
{a.badr,k.brown}@4c.ucc.ie
[2] EMN/LINA UMR CNRS 6241, CIRRELT
arnaud.malapert@emn.fr

Abstract. Effective management of maintenance in buildings can have a significant impact on the total life cycle costs and on the building energy use. Nevertheless, the building maintenance scheduling problem has been infrequently studied. In this paper, we present constraint-based scheduling models for the building maintenance scheduling problem, where each activity has a set of alternative resources. We consider two different models, one using basic constraints, and the other using our new and modified global constraints, which handle alternative disjunctive resources for each activity to allow propagation before activities are assigned to resources. We evaluate these models on randomly generated problems and show that while the basic model is faster on smaller problems, the global constraint model scales better.

Keywords: Building maintenance scheduling, optional activities, alternative resources, disjunctive global constraint

1 Introduction

The operation of large-scale buildings is a significant commercial cost. Suboptimal performance of heating and ventilation systems wastes energy and impacts on the productivity of occupants. Maintenance for large buildings is estimated to consume each year 2% of the total replacement cost of the building [2], with improved maintenance offering up to 20% reductions in the annual energy cost [3]. Effective management of the maintenance process involves solving a scheduling problem with multiple maintenance engineers, interrelated activities, and complex costs on resource use and delays of activities.

Constraint Programming (CP) is one of the most successful techniques for solving scheduling problems with much of its success based on the use of global scheduling constraints, which enable efficient propagation based on commonly occurring problem substructures. In many practical problems, there are multiple discrete resources, and each activity may be executed by any one of a subset of those resources. However, the most successful constraint filtering techniques assume that resources are already assigned to activities before problem solving begins. Some global constraints do model resources where activities are optional,

but these do not take account of the fact that each activity must eventually be scheduled on some resource. In this paper, we consider the problem of building maintenance scheduling. We develop two different models, one using simple constraints, and the other using global constraints representing the alternative resources and optional activities. In the latter model , we use a modified version of the disjunctive global constraint and a new global constraint useResources, which enables modelling resource requirements of activities, i.e., use k of n resources. Finally, we evaluate the two different models and implementations on randomly generated building maintenance scheduling problems. We show that while the basic model is faster on smaller problems, the global constraint model scales better.

The rest of this paper is organized as follows: Section 2 gives description for the application problem, while Section 3 provides the necessary background for the work presented in this paper. Models developed for the building maintenance scheduling problem are discussed in Section 4 and the specification of our global constraints is presented in Section 5. Section 6 provides the experimental evaluation. Finally, Section 7 concludes with a summary and outline for future work.

2 Problem Description

The building maintenance scheduling problem involves a set of maintenance activities, which must be carried out by a set of maintenance engineers with the aim to assign for every activity a start time and an engineer such that all constraints are satisfied and the associated operating costs are minimized. Generally, maintenance activities are either *planned* or *reactive*. Planned maintenance is carried out at predetermined time intervals to prevent degradation or failure of building components and systems, while reactive maintenance is carried out at short notice in response to reported faults, thus making the scheduling problem dynamic. *Emergency* maintenance is a special case of reactive maintenance that causes threat to health and safety and is usually related to a gas (e.g., gas leak), electricity, or water problem.

Typically, a maintenance schedule should satisfy a variety of constraints. For example, some constraints define inter-dependency relations that might occur between activities including *concurrency*, *precedence*, and *non-overlapping*. Concurrency constraints may require a set of activities to be conducted at the same time, while precedence constraints ensure that one set of activities should conclude before another set starts. Non-overlapping constraints restrict a set of activities from overlapping during their execution, to ensure continued safe operation. Additionally, some activities have required skills, and thus can only be carried out by a qualified staff. Furthermore, maintenance involves disruption to the building occupants, and thus activities may be restricted to specified time windows, which might differ over the scheduling horizon. For example, one activity might have the time window (Mon - Wed, 09:00 - 12:00), while another activity could only be executed during one of the following intervals: (Tues, 11:00

- 14:00), or (Thur, 15:00 - 17:00). Hence, activities might have irregular time windows. Finally, engineers execute one activity at a time and have specified skills, limits on their working hours, and windows on their availability, e.g., engineers cannot be assigned activities during their lunch breaks.

An optimized building maintenance schedule is one that satisfies all constraints while minimizing the associated operating costs including maintenance and energy costs. Energy costs are associated with the delayed scheduling of maintenance for components consuming excess amounts of energy if they are faulty, e.g., heaters. Maintenance costs correspond to cost overheads as a result of scheduling activities out of contracted working hours and penalties due to scheduling activities beyond their contracted response time windows. A response time window states when an activity should be carried out according to a service level agreement (SLA). Typically, the SLA defines an expected minimal level of service and may specify financial penalties for the service provider in case the service falls below the minimal level, e.g., failing to address a serious failure in a timely manner. Reactive maintenance may have energy cost associated with it, while planned and reactive maintenance may both incur maintenance costs. We consider scheduling activities within a time-frame of one week.

3 Background

Building maintenance scheduling problems have been infrequently studied in the literature. To the best of our knowledge, this problem is investigated in only one paper [11]. It presents a building maintenance system that uses RFID in maintenance management and includes a scheduling module. This module uses mathematical programming in scheduling planned maintenance activities, while minimizing the total maintenance time. The scheduling module, however, provides schedules for planned maintenance only. Additionally, no details are given regarding problem modelling and handling of different constraints including restrictions of unary resources and temporal constraints.

Constraint-based scheduling models typically involve rich representations of both activities and resources. Activities are usually modelled using three variables representing their start, end, and duration, with $[est, lct]$ representing the time window in which an activity executes, where est and lct are the activity earliest start and latest completion times, respectively. A coherent activity time window is one that satisfies $lct - est \geq$ minimum duration. Disjunctive global constraints are usually used to model unary resources (i.e., with unit capacity) [5]. This constraint ensures that non-interruptible activities executing over a unary resource are sequenced such that they do not overlap at any time interval. Several filtering algorithms are used in the disjunctive constraint including Edge Finding [8], Not-First/Not-Last [18], Detectable Precedence [18], and Overload Checking [19]. We will call these algorithms the *disjunctive resource filtering algorithms*.

The disjunctive resource filtering algorithms assume that all activities to be scheduled on the resource are known in advance, but in many problems each

activity has a choice of resources, and thus the set of activities allocated to each resource is not fixed until those choices are made. One possible way to handle alternatives is by modelling *optional* activities as proposed by Beck and Fox [6]. Optional activities can be classified into two categories: schedule-based, in which an activity may be omitted from the final schedule (e.g. due to different plans being selected), and resource-based, in which the activity is optional for the resource, but must be included on some resource in the final schedule (e.g. courses allocated to a classroom in a timetable). For both categories, we can maintain a binary allocation variable $e_i^r = \{0, 1\}$ for each activity i and resource r. When $e_i^r = 1$, then activity i is allocated to resource r and is said to be *regular* over that resource; when e_i^r is not yet assigned a value, i is *optional* over r; finally, an activity i is *disabled* over r if e_i^r is set to zero. We focus in this paper on resource-based optional activities.

Some researchers have studied scheduling problems with alternative resources. In [9], Focacci *et al.* presented a general job shop scheduling problem with sequence dependent setup times and alternative resources. They use a constraint called *alternative resources* to apply reasoning over activities with alternative resources. A modified version of this constraint is also presented in [19] to solve scheduling problems with alternative resources. Additionally, some disjunctive resource filtering algorithms have been extended to accommodate reasoning over optional activities. In [17], Vilim *et al.* proposed extensions to detectable precedence, not-first/not-last, and overload checking with time complexity $O(nlogn)$. These algorithms use efficient special data structures, called $\Theta - \lambda$ trees that allow "what if" reasoning over different sets of activities. Furthermore, Kuhnert proposed in [12] an extended version of edge finding that handles optional activities with time complexity $O(n^2)$. In these extended versions, an optional activity is disabled over a resource if it causes its overloading, i.e., no feasible sequencing for activities over the resource exists as activities cannot be processed within their time windows. Additionally, the extended reasoning enables regular activities to modify the time windows of other regular and optional activities over the resource. However, the opposite is not true as optional activities could later be disabled during problem solving. These extended algorithms apply to resource-based and schedule-based optional activities.

In [19], the alternative resources constraint considers a set of T activities and R resources, with the following filtering rules:

$$\forall t \in T, \forall r \in R_t : \; S_t^r \; \cap \; S_t \; = \emptyset \Rightarrow R_t := R_t \setminus \{r\}, \tag{1}$$

$$\forall t \in T, \forall r \in R_t : \; S_t^r \; \cap \; S_t \; \neq \emptyset \Rightarrow S_t^r := S_t^r \; \cap \; S_t \tag{2}$$

$$\forall t \in T : \; S_t := S_t \cap (\bigcup_{(r \in R_t)} S_t^r) \tag{3}$$

For every activity t, S_t represents a non-empty finite of potential start times, while R_t is a non-empty finite set of alternative resources. Additionally, for each

activity $t \in T$ and every alternative resource $r \in R_t$, a non empty finite set of alternative start times $S_t^r := S_t$ is defined.

The aforementioned rules are generally triggered whenever S_t or S_t^r is changed. In rule (1), if no possible start times for an activity t are left over a resource r, then r is removed from the set of alternative resources. In (2), modifications applied to the main possible start times are propagated to its alternatives. Finally, rule (3) propagates updates applied over alternative possible start times to the main possible start times whenever one of these alternatives (i.e., S_t^r) is updated. Throughout this paper, we will refer to these rules by the *alternative resources filtering rules* and use ($\Rightarrow$) in their procedural specification, while using ($\rightarrow$) and ($\leftrightarrow$) in logical constraints.

The scheduling problem with alternative resources is solved in [19] using a number of scheduling constraints including the disjunctive constraint, a sweeping algorithm, which performs global overload checking, and the alternative resource constraint. However, the sweeping algorithm is computationally expensive; its time complexity is $O(n^4)$. Additionally, the disjunctive constraint used in [19] and the scheduling constraints used in [9] do not use the extended disjunctive resource filtering algorithms proposed in [17,12]. Furthermore, to the best of our knowledge, no previous work has investigated constraint-based modelling of resource requirements for activities that are allocated a set of resources selected among alternatives.

4 Building Maintenance Scheduling Models

In this section, we discuss the models developed for the building maintenance scheduling problem. We first start with the basic model in Section 4.1, followed by the global constraint model in Section 4.2. In both models, we handle multi-objective optimization through the *weighted objectives* technique. Weighted objectives combines different objective functions into a single one through a linear weighted summation of these objectives. Scalar weights of objectives are specified according to their relative importance [15].

4.1 The Basic Model

This model is based on [4]. The building maintenance scheduling problem consists of a set of activities $T = \{1....n\}$ and a set of engineers $R = \{1....m\}$. For each activity $i \in T$, we define the following variables: $start_i$ is the start time for the activity, with a domain of integers ensuring time windows are obeyed; d_i is the activity duration [3]; end_i is the end time for the activity, with a domain of integers ensuring time windows are obeyed; c_i is the assigned engineer, with a domain of integers, subset from R, ensuring that only engineers with the required skills can be assigned; x_i is the cost resulting from scheduling activity i

[3] Durations of activities are generally considered as variables. In our problem instances, however, we assume activity duration to be a constant integer regardless of which engineer is assigned the activity.

at its chosen start time; s_i is an auxiliary variable linking the start time to an array of costs. Finally, a variable p represents the total cost of the schedule.

Constants included in the model for each activity i are as follows: dt_i is an array of start times such that each value in the domain of $start_i$ appears exactly once in this array; $tcost_i$ is an array of costs, of the same length as dt_i, representing the cost of each start time.

For each activity $i \in T$, we define the following constraint:

$$start_i + d_i = end_i \tag{4}$$

Since the start time windows and costs in building maintenance scheduling are irregular, we use for each activity i the two parallel arrays dt_i and $tcost_i$ such that if $dt_i[index] = t_s$, where t_s is the start time value and $index$ is its position in the array , then the cost of starting activity i at time t_s is $tcost_i[index]$. We represent this using the two element constraints:

$$element(s_i, dt_i, start_i) \tag{5}$$

$$element(s_i, tcost_i, x_i) \tag{6}$$

The $element(I, S, X)$ constraint states that $S[I] = X$. Additionally, for each pair of activities i and j, such that $i \neq j$, we define the disjunctive constraint which ensures that two activities cannot be executed at the same time by the same engineer:

$$c_i = c_j \rightarrow (end_i \leq start_j) \vee (end_j \leq start_i) \tag{7}$$

Depending on the problem instance, we also add constraints of the following types:

$$Concurrency: \ start_i = start_j \tag{8}$$

$$Precedence: \ end_i \leq start_j \tag{9}$$

$$Non-overlapping: \ (end_i \leq start_j) \vee (end_j \leq start_i) \tag{10}$$

$$\forall S, \ AllDifferent(\{c_i : \ i \in S\}) \tag{11}$$

The first three constraint types represent inter-dependency relations that might occur between activities. Additionally, for each maximal set S of concurrent activities, the redundant constraint in (11) is defined to restrict concurrent activities to be assigned to different engineers. We then specify total cost of the schedule as:

$$p = \sum_{i \in T} x_i \tag{12}$$

Finally, the objective function is to minimize p.

4.2 The Global Constraint Model

In addition to the basic model above, we consider another model, which uses global constraints to capture known relationships between activities and resources, in an attempt to improve propagation. Firstly, we add the allocation variables. For each activity $i \in T$ and for every engineer $r \in R$, we define a binary variable e_i^r, to indicate whether the activity i is assigned to engineer r. If engineer r is not qualified for the activity (i.e., is not among the set of alternative engineers with the required skills), e_i^r is set to zero. This requires a channeling constraint (13), which links c_i with the e_i^r:

$$\forall i \in T, \forall r \in R: \; c_i = r \leftrightarrow e_i^r = 1 \tag{13}$$

Secondly, we remove all constraints of type (7), which only propagate once activities have been assigned to engineers, and instead add two global constraints:

$$\forall r \in R: \; AltDisj([start_1, .., start_n], [d_1, .., d_n], [end_1, .., end_n], [e_1^r, .., e_T^r]) \tag{14}$$

$$\forall i \in T : useResources(1, \; start_i, \; d_i, \; end_i, \; [e_i^1, ...e_i^m], \; R) \tag{15}$$

For each engineer, the alternative disjunctive constraint (14) lists all activities which could possibly be executed by that engineer, and ensures that if they are assigned to the engineer then they do not overlap. Additionally, the useResources constraint (15) ensures that each activity must be eventually assigned to exactly one engineer. We use two separate global constraints that reason about alternative resources to accommodate the use of hypothetical domains and enable modelling heterogeneous resource requirements as explained in Section 5.

Finally, we add a cumulative constraint (16) to enable early propagation over the domains of start time variables, where h_i is a unit height for activity i. As little pruning can take place until activities are assigned a resource, it is possible that multiple activities could be assigned the same start time during search even though they are sharing some resources. Hence, the cumulative constraint restricts the number of activities taking the same start time to be at most equal to the total number of engineers.

$$cumulative([start_1, ...start_T], [d_1, ..., d_T], [end_1...., end_T], [h_1, .., h_T], \; m) \tag{16}$$

5 The Global Constraints

In this section, we present our modifications to the alternative disjunctive constraint (the one that includes the extended disjunctive resource filtering algorithms) and introduce our new global constraint useResources. Note that these constraints are not specific to our building maintenance scheduling problem, and can be applied to any scheduling problem involving alternative resources.

5.1 The Alternative Disjunctive Constraint

Scheduling problems with alternative resources can be defined as follows:

Definition 1 *"There is a set of activities* $T = \{1....n\}$ *and a set of resources* $R = \{1.....m\}$. *Every activity* $i \in T$ *has a set of alternative resources* $A \subset R$, *where* $|A| \geqslant 1$. *Furthermore, for every* $i \in T$ *and for every* $r \in R$, *a boolean variable* e_i^r *is defined to reflect the status of activity* i *over resource* r. *We must allocate for every activity* i *a resource* r *such that:* $\forall i, j \in T, \forall r \in R : (e_i^r = 1) \wedge (e_j^r = 1) \rightarrow (end_i \leq start_j) \vee (end_j \leq start_i)$*".*

Alternative resources filtering rules discussed in [19] do not use binary allocation variables to represent the activity status over its alternative resources. These binary variables are commonly used to represent alternatives as shown in Section 3. They are also used in the extended disjunctive resource filtering algorithms proposed in [17,12]. Hence, we modify the alternative resources filtering rules to enable an easy integration with the extended disjunctive resource filtering algorithms and to ensure compatibility with the useResources constraint discussed in Section 5.2.

Considered from the resource point of view, a subset of the activities may be scheduled on the resource. This leads to the following definition for the alternative disjunctive constraint : $AltDisj([start_1, .., start_n], [d_1, .., d_n], [end_1, .., end_n], [e_1^r, .., e_n^r])$. The straightforward specification for the constraint would require, for each possible activity on the resource, a new variable $start_i^r$ representing the possible start times for the activity on that resource. However, this creates two problems. First, if filtering on the resource removes all possible start times for the optional activity, that creates an empty domain, which will cause backtracking, even though no inconsistency has been discovered. Instead, an optional activity should be disabled over the resource, and thus the internal solver behavior should be modified. Additionally, introducing these alternative variables and domains modifies the constraint network, and thus interferes with reformulation methods and search heuristics, particularly those based on degree.

To overcome these drawbacks, we introduce hypothetical domains. A hypothetical domain Dh_i^r represents the time window of an optional activity i over an alternative resource r. Initially, $Dh_i^r = Dm_i = [est_i, lct_i]$. Dm_i represents bounds for activity i main time window, which reflects a unified view for different time windows available over the activity alternative resources. We use bounds representation for hypothetical domains rather than a domain of values as the extended disjunctive resource filtering algorithms apply bound consistency reasoning. This representation also enables reasoning on variable durations.

The use of hypothetical domains limits the number of variables that need to be created. Secondly, when an activity becomes regular over a resource, its hypothetical time window is reflected over the main time window (i.e., $e_i^r = 1 \Rightarrow Dm_i := Dm_i \cap Dh_i^r$) and the alternative disjunctive constraint references the activity main time window instead of its hypothetical one. Hence, propagation on a resource filters the hypothetical domain and transfer to the main domain once the activity becomes regular. Finally, when a hypothetical domain becomes

empty or incoherent, the CP solver sets the corresponding binary allocation variable to zero.

We modify the alternative resources filtering rules to enable using hypothetical domains and binary allocation variables. The first two rules are redefined as follows:

$$\forall i \in T:\ e_i^r = \{0,1\} \wedge\ |Dh_i^r \cap Dm_i| < min(d_i) \Rightarrow e_i^r := 0 \tag{17}$$

$$\forall i \in T:\ e_i^r = \{0,1\} \wedge\ |Dh_i^r \cap Dm_i| > min(d_i) \Rightarrow Dh_i^r := Dh_i^r \cap\ Dm_i \tag{18}$$

These action rules are triggered whenever the domain of one of the activity variables (start, end, or duration) is modified, initialized or values are removed from its domain, and thus the main time window is updated. They are also triggered when a hypothetical domain is updated by the extended disjunctive resource filtering algorithms, where $min(d_i)$ is the minimum duration of activity i. For example, if the domain of an activity related variable is reduced to a singleton, rule (18) updates the hypothetical domain. After that, rule (17) is checked to ensure that the time window for the updated hypothetical domain is still consistent and if this is not the case, then the activity is disabled over its alternative resource. To illustrate, we assume having an activity 1 with variables $start_1 = [5{-}9]$, $end_1 = [7{-}11]$, and $d_1 = 2$. Hence, $Dm_1 = [5{-}11]$. Additionally, we assume that this activity has two alternative resources and its hypothetical domain over the first resource is $Dh_1^1 = [5-11]$ and is $Dh_1^2 = [8-11]$ over the second resource. During problem solving, if $start_1$ is assigned the value 5, then activity 1 is disabled over the alternative resource 2. Note that when values are removed from the domain of an activity related variable, a hypothetical domain is updated only if the removed value contributes to its boundaries. Additionally, the extended disjunctive resource filtering algorithms are triggered as normal when the hypothetical domains and/or main domains are modified.

5.2 Allocation Constraint: useResources

The representation of alternative resources through binary variables mandates the use of constraints to ensure that resource requirements are satisfied. Additionally, an activity might have different time windows over different resources, and thus it is necessary to ensure that their updates are propagated back to the activity main time window. Typically, every resource maintains hypothetical domains corresponding to its optional activities and it does not retain any information related to other hypothetical domains for these activities over other alternative resources. Additionally, the alternative resources filtering rule (3) could be generalized to represent heterogeneous resource requirements such that an activity could be allocated to k resources selected among a given subset of alternatives instead of one resource. Hence, to simplify problem modelling and gain

in flexibility, this rule is defined in a separate new global constraint called useResources. Therefore, an activity could specify different resource requirements like for example a demand for k_1 workers and k_2 machines.

The constraint is defined for an activity i and a requirement $1 \leq k \leq m$ as follows: useResources(k, $start_i$,d_i, end_i, $[e_i^1, ...e_i^m]$, R). This constraint integrates the boolean sum constraint $\sum_{j=1..m} e_i^j = k$ or $\sum_{j=1..m} e_i^j \geq k$ defined over the binary allocation variables. Additionally, it includes the filtering rule that aggregates deductions emerging from alternative time windows until resource requirement is satisfied. Let $\min_P(k, val_i^r)$ be the k-th minimum of val_i^r among a subset P from the set of resources R. For example, the third minimum of the set $\{1, 2, 2, 3\}$ is 2. Rule (19) is the modified version of rule (3) and is applied when a single resource is needed. Rule (20) is used when it is required to allocate k resources.

$$\forall i \in T,\ k = 1 :\ Dm_i := Dm_i \cap \bigcup_{1 \in e_i^r} Dh_i^r \tag{19}$$

$$\forall i \in T,\ k > 1 :\ Dm_i := Dm_i \cap \left[\min_{1 \in e_i^r}(k, est_i^r), \max_{1 \in e_i^r}(k, lct_i^r)\right] \tag{20}$$

In addition, the framework offers a great flexibility since many types of resource requirements could be integrated by defining new constraints interacting with hypothetical domains.

6 Experimental Evaluation

We have developed the presented models in Choco [13]; an open source Java constraint solver, which provides access to recent implementations for a wide range of global constraints. The flexibility of its scheduling component reduces the implementation effort, while providing satisfactory results in solving job-shop and open-shop scheduling problems as shown in the fourth international CSP solver competition [1] and it demonstrates a satisfactory performance with dedicated approaches as shown in [14,10]. The basic model is relatively straightforward, and maps exactly onto Choco primitives. The global constraint model, however, uses two global constraints. The alternative disjunctive constraint is already available in Choco, but without hypothetical domains, the integrated alternative resources filtering rules presented in Section 5.1, and no filtering is carried out, by the extended disjunctive resource filtering algorithms, over the time window of an optional activity until it becomes regular. We implement the modified alternative disjunctive constraint, the useResources constraint, and the extended edge finding filtering algorithm [12].

The extended disjunctive resource filtering algorithms are not idempotent. They are executed in iterations in the alternative disjunctive constraint till no further changes are found by any filtering algorithm, i.e., a fixed point is reached. The order in which these algorithms are called is the one discussed in [14].

This order reduces the total solving time, with no effect on the resulting fixed point, by minimizing the number of sorts required by the data structures. Our experiments show that the extended edge finding algorithm is computationally expensive. Hence we exclude this algorithm in our experimental evaluation.

In our experiments, we consider (M_1) to be the global constraint model described in Section 4.2, while (M_2) is the basic model illustrated in Section 4.1. In both models, we use the variable ordering heuristic ModReg that uses cost based reasoning to guide the search process towards the most viable course [4], based on the Regret heuristic [16]. The regret is the additional cost to be paid for a variable assignment over the cost lower bound if the variable is not assigned the suggested value at its lower bound. The regret heuristic suggests assigning the variable with the highest regret first so as to minimize the risk of paying a higher cost if the best assignment becomes infeasible. In the context of the maintenance scheduling problem, regret is calculated as the difference in cost between the earliest and second earliest start times. Additionally, the definition of the regret heuristic is extended in [4] by breaking ties through selecting the variable corresponding to the activity with the minimum number of time slots with zero cost or with minimum duration. This heuristic gives the best performance in maintenance scheduling in [4]. The branching strategy used in our models assigns for each selected activity, based on ModReg, the start time variable followed by the engineer variable. Thus, domains of decision variables in both models are enumerated such that the engineer with the earliest starting time is assigned first.

We built a generator that randomly generates test cases with different problem sizes and constraint relations, where problem size is the number of activities in each test case. We depend on randomly generated instances as energy information is not considered in real problems where a conventional first come first serve scheduling mechanism is usually followed. However, the data used in the generator depends on the information gathered from discussions with our collaborators [4].

In the random generator, we use a fixed number of engineers (15). The number of alternative engineers for each activity is generated randomly between one and four. Their identifiers are also generated randomly between one and fifteen. Activities participating in a dependency relation and those that are independent have distributions shown in Table 1a in columns (dep) and (indep), respectively. Additionally, the number of activities that could participate in a dependency relation is generated randomly between two and four and the inter-dependency relation type is also generated randomly.

Available time windows for activity execution are generated through two separate operations: the selection of days and selection of time frames within these days. First, we generate randomly for every activity up to two separate ranges of days per week and up to two separate time frames within every generated range of days. For a single range of days, we choose either all five days Mon-Fri

[4] Industrial partner Spokesoft, Civil Engineers, and the Building and Estates office at University College Cork

(with probability 0.6), or a random sub-interval of Mon-Sun (with probability 0.4). If two separate ranges of days are required, then we generate randomly two non-overlapping sub-intervals of Mon-Sun. One time frame is generated with distributions 0.6 for a 09:00 - 17:00 time frame and 0.4 for a time frame that is generated randomly between 09:00 - 20:00. The two separate time frames involve the random generation of two non-overlapping time frames between 09:00 - 20:00.

The activity maintenance type (planned (plan), reactive (reac), and emergency (emerg)) is generated with the distributions presented in Table 1a. The response time window is generated randomly depending on the maintenance type; a four hours response window is assumed for emergency, 1, 2, 3, 7, or 15 days for non-emergency reactive maintenance, and a period of up to 30 days for planned maintenance. Note that, for response time windows greater than one week, we consider only the part of the response time window, which is included in the scheduling horizon. Expected durations for activities are generated with distributions shown in Table 1a, where slot granularity is assumed to be 30 minutes. The excess amount of energy, if any, consumed by the malfunctioning component per unit time is generated with distributions representing no excess energy, low, medium or high energy levels as shown in Table 1b. Finally, missed window refers to the time window already passed from the activity response time before its scheduling. The distribution of activities having missed windows and those that do not are shown in Table 1b. Missed window is used to specify the earliest time at which a penalty applies and its size is generated randomly; for non-emergency reactive maintenance it includes up to 15 days and up to 30 days for planned maintenance. We also consider only the part of the missed window which is included within the scheduling time frame.

We test the performance of the presented models using test cases with different problem sizes, and we generate randomly a set of ten scenarios for each problem size. To avoid extended search time, we set a solving time limit of 10 minutes. Tests were carried out using a 2.40 GHz Intel Core 2 Duo PC with 3.48 GB of memory and running Windows XP. Table 2 illustrates experimental results for test cases that include 20, 50, 70, 100, and 120 activities. Columns Avg.FT, Avg.LT, and Avg.TP give the average times in seconds to find the first solution, the best solution, and prove optimality, respectively. Additionally, column Avg.BT gives the average number of backtracks, while Avg.Nd gives the average number of nodes. This table shows that M_2 is generally much faster in comparison to M_1 (only slower in problems with size 100 for the time required to prove optimality and those with size 120 for the time taken to find the best solution). As expected, the embedded reasoning in the global constraints in M_1 significantly reduces the number of nodes and backtracks.

The branching strategy used in the assignment of the start time and engineer variables enables an efficient use of the global constraints in M_1. This is because after an activity is assigned a start time, the alternative disjunctive constraint disables alternative engineers who cannot carry out the activity with its new time window. To study the impact of this strategy, we experimented with a different

branching strategy that assigns first the start time variables for all activities according to the ModReg heuristic, then assigns engineer variables based on the Dom/WDeg heuristic [7]. Results show that using this strategy, the solver failed to provide any solutions for the majority of the test cases within the specified time limit due to incorrect decisions for the assignment of start times taken at the top of the search tree.

To better understand the effect of the global constraints on the model solving ability, we test the two models, using the same settings, over larger problem instances (150, 200, 220, and 240). Table 3 shows the percentages of test cases solved to optimality and those with solutions found within the aforementioned time limit. This table shows that reasoning incorporated in these constraints significantly improves the model solving ability; it enables M_1 to solve all the test cases and prove solution cost optimality for a few of them (20% of test cases with size 150). However, M_2 shows a degrading performance as the problem size increases. These results demonstrate the effect of problem structure exploitation on solver performance when the maintenance scheduling problem becomes more complex.

Table 1: The probability distribution of features in the test case generator

(a) Dependency, duration, and maintenance type

Feature	Dependency		Duration (Slots)				Maintenance Type		
Value	0: dep	1: indep	1	2	3	4	1: plan	2: reac	3: emerg
Prob.	0.55	0.45	0.50	0.30	0.10	0.10	0.50	0.40	0.10

(b) Energy consumption and missed window

Feature	Energy Consumption				Missed Window	
Value	0:no Energy	1:Low	2:Medium	3:High	0:no missed	1: with missed
Prob.	0.45	0.35	0.15	0.05	0.80	0.20

Table 2: Experimental results for problems that include up to 120 activities in the two CP models

Problem	**Avg.FT**		**Avg.LT**		**Avg.TP**		**Avg.BT**		**Avg.Nd**	
Size	**M_1**	**M_2**	**M_1**	**M_2**	**M_1**	**M_2**	**M_1**	**M_2**	**M_1**	**M_2**
20	0.071	0.027	0.091	0.027	0.119	0.038	**0**	0.2	36.8	36.8
50	0.511	0.110	0.828	0.128	0.879	0.156	**0.2**	6.3	129.1	129.1
70	1.126	0.276	2.004	0.355	2.081	0.415	**0.7**	24.2	184.6	184.6
100	2.649	0.859	103.949	40.828	**135.360**	161.572	**2489.6**	113335	**2650.6**	61194
120	4.019	2.194	**23.284**	43.425	144.436	136.458	**3439.4**	23480.1	**3058.6**	16235.6

Table 3: Percentages of test cases with proven optimal and non-optimal solutions

	optimal (%)		With Sol(%)	
Problem Size	**M_1**	**M_2**	**M_1**	**M_2**
150	20.0	10.0	100.0	80.0
200	0.0	0.0	100.0	70.0
220	0.0	0.0	100.0	30.0
240	0.0	0.0	100.0	20.0

7 Conclusions

We have investigated a building maintenance scheduling problem. We have developed a constraint model that involves basic constraints, and another model that uses global constraints to represent activities with alternative resources with the aim to allow propagation before resources are allocated. We also present our modifications to the alternative disjunctive constraint and introduce our global constraint useResources that enables modelling resource requirements. Experimental evaluation of these models shows that the basic model is faster on smaller problems, but as the problem size increases, the global constraint model scales better, and continues to produce solutions when the simple model does not.

In future work, we will conduct more experiments to precisely evaluate the performance of the modified/new global constraints using benchmark scheduling problems. These constraints constitute an appealing approach to solving a wide range of practical scheduling problems including multi-processor task scheduling. We will also focus on investigating a new three stage search strategy inspired from scheduling and packing problems. The first stage determines the relative ordering between pairs of activities linked by non-overlapping constraints, while the second stage (inspired by ModReg and packing heuristics) restricts start time windows of activities before allocating an engineer. This restriction aims to efficiently utilize the alternative disjunctive and useResources constraints. Finally, the third stage assigns starting times to activities. The existence of a solution is very likely in the third stage, since time windows of activities are tight and consistent. Finally, we consider improving the constraint model by the reformulation and deduction of redundant constraints.

Acknowledgments

This work is part of the ITOBO project, funded by Science Foundation Ireland under grant No.07.SRC.I1170. The authors would like to thank École des Mines de Nantes and École Polytechnique de Montréal; Anika Schumann for her valuable comments on a draft of the paper; and Ena Tobin, the Building and Estates office at University College Cork, and our industrial partner Spokesoft for the information provided about maintenance scheduling.

References

1. Fourth international csp solver competition. http://www.cril.univ-artois.fr/CPAI09/.
2. The whole building design guide. National Institute of Building Sciences, 2005.
3. Energy efficiency in buildings. World Business Council for Sustainable Development, 2008.
4. Aliaa M. Badr and Kenneth N. Brown. Building maintenance scheduling using cost-based reasoning and constraint programming techniques. In *8th European Conference on Product and Process Modelling (ECPPM -10)*, September 2010.
5. Philippe Baptiste and Claude Le Pape. Disjunctive constraints for manufacturing scheduling: Principles and extensions. In *In Proceedings of the Third International Conference on Computer Integrated Manufacturing*, 1995.
6. J. Christopher Beck and Mark S. Fox. Constraint-directed techniques for scheduling alternative activities. *Artif. Intell.*, 121(1-2):211–250, 2000.
7. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'*, pages 146–150, 2004.
8. J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78(2):146–161, 1994.
9. W. Nuijten F. Focacci, P. Laborie. Solving scheduling problems with setup times and alternative resources. In *4th International Conference on AI Planning and Scheduling*, pages 92 – 101, 2000.
10. Diarmuid Grimes, Emmanuel Hebrard, and Arnaud Malapert. Closing the open shop: Contradicting conventional wisdom. In *Principles and Practice of Constraint Programming*, pages 400–408, 2009.
11. Chien-Ho Ko. Rfid-based building maintenance system. *Automation in Construction*, 18(3):275 – 284, 2009.
12. Sebastian Kuhnert. Efficient edge-finding on unary resources with optional activities. Proceedings of INAP 2007 and WLP 2007 (2007).
13. F. Laburthe and N. Jussien. Choco constraint programming system. http://www.emn.fr/z-info/choco-solver/tex/choco.pdf, February 2010.
14. Arnaud Malapert, Hadrien Cambazard, Christelle Guéret, Narendra Jussien, André Langevin, and Louis-Martin Rousseau. An optimal constraint programming approach to solve the open-shop problem. *Journal of Computing*, (CIRRELT-2009-25), 2009.
15. R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26:369–395, 2004.
16. Michela Milano. *Constraint and Integer Programming: Toward a Unified Methodology*. 2003.
17. Petr Vilím, Roman Barták, and Ondřej Čepek. Extension of o(n log n) filtering algorithms for the unary resource constraint to optional activities. *Constraints*, 10(4):403–425, 2005.
18. Petr Vilím. O(n log n) filtering algorithms for unary resource constraint. In *Proceedings of CP-AI-OR 2004*, pages 335–347. Springer-Verlag, 2004.
19. Armin Wolf and Hans Schlenker. Realising the alternative resources constraint. In *INAP/WLP*, pages 185–199, 2004.