Computing Tight Time Windows for RCPSPWET with the Primal-Dual Method*

András Kéri[1] and Tamás Kis[2,**]

[1] Chair of Business Administration, Transport and Logistics, Technical University of Dresden, Andreas Schubert str. 23, D-01069 Dresden, Germany
andras.keri@mailbox.tu-dresden.de

[2] Computer and Automation Institute, Kende str. 13-17, H-1111 Budapest, Hungary
tamas.kis@sztaki.hu

Abstract. In this paper we combine OR and CP techniques to solve the Resource-Constrained Project Scheduling Problem with Earliness-Tardiness costs and general temporal constraints. Namely, we modify the Primal-Dual algorithm for solving the maximum-cost flow problem in a network to deduce tight time windows for activities with respect to a finite upper bound on the optimal objective function value. We compare our method to the only exact method in the literature. Our results show that time window computations and additional domain filtering techniques may improve the performance of tree-search based methods.

1 Introduction

Suppose we have n activities $V = \{1, \ldots, n\}$ with given *processing time* $p_j \geq 0$, *due-date* $d_j \geq 0$, *unit earliness cost* $e_j \geq 0$ and *unit tardiness cost* $t_j \geq 0$ for each $j \in V$. Furthermore, there is a set $A \subset V \times V$ of pairwise temporal relations between the activities with time lags $\delta_{i,j}$, $(i,j) \in A$, where the $\delta_{i,j}$ can be 0, positive or negative, which enables the modeling of minimum and maximum time lags. In addition, there is a finite set R of resources to carry out the activities. Each resource $k \in R$ has a finite capacity $b_k > 0$, and for each activity $i \in V$, its resource requirements are given by non-negative numbers $r_{i,k}$, $k \in R$. All the problem data are integral.

We have to determine the non-negative *start time* S_j of each activity j subject to (i) temporal constraints: $S_j - S_i \geq \delta_{i,j}$, $\forall (i,j) \in A$, and (ii) resource constraints: $\sum_{i \in \mathcal{A}(S,t)} r_{i,k} \leq b_k$, for each $k \in R$ and any time point t, where $\mathcal{A}(S,t)$ denotes the set of those activities that are being executed at time point t, i.e., $\mathcal{A}(S,t) = \{i \in V \mid S_i \leq t < S_i + p_i\}$. We say that activity j is *early* if $S_j + p_j < d_j$ and *tardy* if $S_j + p_j > d_j$. The cost incurred when activity j is early is $e_j \cdot \max\{0, -S_j + d_j - p_j\}$, while the cost of being late is $t_j \cdot \max\{0, S_j - d_j + p_j\}$.

* The research of Tamás Kis has been supported by the János Bolyai Research Grant No. BO/00380/05 and by the NKFP Grant No. 2/010/2004 (VITAL).

** Corresponding author.

The objective is to minimize the total earliness-tardiness cost over all activities. To summarize, the *Resource-Constrained Project Scheduling Problem with Earliness-Tardiness costs (RCPSPWET)* studied in this paper can be stated as follows:

$$\min \sum_{j \in V} (e_j \cdot \max\{0, -S_j + d_j - p_j\} + t_j \cdot \max\{0, S_j - d_j + p_j\})$$

subject to the constraints (i) and (ii). This problem is NP-hard in general and becomes polynomial if there are no resource constraints [21].

The objective function is *non-regular*, as it is not monotone in the starting (or completion) times of the activities. A classification of non-regular objective functions can be found in [15].

We are aware of only one exact method for RCPSPWET [18] with general temporal constraints, which is nevertheless similar to that of [14] for the maximum net present value problem. In fact, almost the same branch and bound procedure can be applied to the two different problems [15]. The lower/upper bounds can be computed by the *steepest ascent method* [18], [19]. In addition, the procedure makes various deductions to cut off a large part of the search tree. This includes the subset dominance rule and inference of new arcs between pairs of resource-disjunct activities. The special case with precedence constraints only (no maximal time lags) has been studied by Vanhoucke et al. [20] and then by Kéri and Kis [9], [10]. In particular, Kéri and Kis has proposed a fast approximation method for determining approximate time windows for activities, but that method seems to work only when there are no directed cycles in the precedence graph. Notice that when maximum time lags are allowed, then cycles of non-positive lengths are allowed. For the job-shop special case Beck and Refalo [3] have proposed various hybrid solution techniques. Their procedures exploit that there are only end-to-start precedence constraints and that all resources are of unit capacity. Some of their procedures (CRS family) also make use of the fact that only the last activities of the jobs affect the cost function directly. From a different perspective, the resource-relaxed problem is a special case of the *Simple Temporal Problem with Preferences*, proposed by Khatib et al. [11]. In that problem, between two events X_i and X_j there can be hard minimum and maximum time lags, or a *preference function* $F_{(X_i,X_j)}(t)$ indicating the preference of scheduling X_j t time units after X_i. For piecewise linear concave preference functions, a solution which maximizes the sum of preferences can be found by solving a linear program as observed by Morris et al. [13]. Kumar [12] has recently noticed that the linear program is equivalent to a minimum cost flow problem in an appropriate network.

Our main contribution is an algorithm for computing tight time windows for the activities using only the best known objective function value. Our method is based on the Primal-Dual method for computing a maximum-cost flow in a network. Time windows are then used for applying additional domain filtering techniques, published in the literature, in order to narrow further the time windows of activities. Narrower time windows permit, on the one hand, to infer more arcs between resource-disjunct activities, and on the other hand to compute stronger lower bounds. As a result, in many cases it suffices to explore a smaller portion of the search tree to find an optimal solution and prove its optimality.

The structure of the paper is as follows: In Sections 2 we summarize known facts about the resource-relaxed problem. In Section 3 we describe the network computations and in particular we provide the method for computing tight time windows. Section 4 sketches the branch and bound procedure that we apply. We do not give all details as these are well documented in the literature. The performance of our method is evaluated in Section 5.

2 Preliminaries

Below we formalize the resource-relaxed problem as a linear program and recognize that it is dual to a maximum cost flow problem. The same conclusion has been reached in [21]. We pay special attention to the modeling of time windows.

We introduce two dummy activities, 0 and $n+1$, and let $V^+ = V \cup \{0, n+1\}$. Let A^+ contain all the temporal constraints from A and the new constraints $(0, j)$ with $\delta_{0,j} = 0$, $(j, 0)$ with $\delta_{j,0} = -\infty$, $(j, n+1)$ with $\delta_{j,n+1} = -d_j + p_j$ and $(n+1, j)$ with $\delta_{n+1,j} = d_j - p_j$ for $j \in V$. Let $A^- = A^+ \setminus \{(j, n+1), (n+1, j) \mid j \in V \cup \{0\}\}$. Moreover, let $e_0 = t_0 = \infty$ and $d_0 = 0$. After these preparations, the linear program modeling the resource-relaxed problem is:

$$v(S) = \min \sum_{j \in V \cup \{0\}} e_j w_{n+1,j} + t_j w_{j,n+1} \tag{1}$$

subject to

$$S_j - S_i \geq \delta_{i,j}, \quad \forall (i,j) \in A^-, \tag{2}$$
$$S_j - S_{n+1} + w_{n+1,j} \geq \delta_{n+1,j}, \quad \forall j \in V \cup \{0\}, \tag{3}$$
$$S_{n+1} - S_j + w_{j,n+1} \geq \delta_{j,n+1}, \quad \forall j \in V \cup \{0\}, \tag{4}$$
$$w_{n+1,j}, w_{j,n+1} \geq 0, \quad \forall j \in V, \tag{5}$$
$$S_{n+1} = 0. \tag{6}$$

Inequalities (2) represent the temporal constraints, while (3) and (4) express the earliness and tardiness of activity j, respectively. Since $e_0 = t_0 = \infty$ and $d_0 = 0$, in any optimal solution $S_0 = 0$. We deliberately do not set S_0 to 0 explicitly to facilitate fast re-optimization as explained in Section 3.1. Then the dual problem is:

$$\max \sum_{(i,j) \in A^+} \delta_{i,j} X_{i,j} \tag{7}$$

subject to

$$\sum_{(j,i) \in A^+} X_{j,i} - \sum_{(i,j) \in A^+} X_{i,j} = 0, \quad \forall i \in V^+, \tag{8}$$

$$X_{n+1,j} \le e_j \quad \forall j \in V \cup \{0\}, \tag{9}$$
$$X_{j,n+1} \le t_j \quad \forall j \in V \cup \{0\}, \tag{10}$$
$$X_{i,j} \ge 0, \quad \forall (i,j) \in A^+. \tag{11}$$

Clearly, this is a maximum cost flow problem in the network $N(\delta) = (V^+, A^+, \delta)$. We will refer to (7)-(11) as the *primal problem* and to (1)-(6) as the *dual problem*. It is known that for non-negative e_j and t_j values, the optimal $w_{j,n+1}$ and $w_{n+1,j}$ values are uniquely determined by the optimal S values by the formulas $w_{n+1,j} = \max\{0, -S_j + d_j - p_j\}$ and $w_{j,n+1} = \max\{0, S_j - d_j + p_j\}$. Therefore, the optimal dual solution can be characterized by the values of the S_j variables.

We recapitulate the complementary slackness optimality conditions for later reference. The *reduced cost* of arc $(i,j) \in A^+$ is defined as $rc_{i,j}(S) = \delta_{i,j} + S_i - S_j$. Let $u_{i,j}$ denote the upper bound on arc (i,j). Notice that $u_{i,j} = \infty$ except on the arcs $(n+1, j)$ and $(j, n+1)$, $j \in V$. It is known that a pair of primal and dual optimal solutions must satisfy the conditions:

If $rc_{i,j}(S) < 0$, then $X_{i,j} = 0$.
If $rc_{i,j}(S) > 0$, then $X_{i,j} = u_{i,j}$.
If $0 < X_{i,j} < u_{i,j}$, then $rc_{i,j}(S) = 0$.

With respect to any $\bar{X} \in \mathbb{R}^{|A^+|}$, define the *excess* of node i as $ex(i) = \sum_{(j,i)\in A^+} \bar{X}_{j,i} - \sum_{(i,j)\in A^+} \bar{X}_{i,j}$. Node i is a *source node* if $ex(i) > 0$, and it is a *sink node* if $ex(i) < 0$.

3 Network Computations

If we relax the resource constraints, we are left with a much easier problem, which can be solved in polynomial time [21]. Various methods have been suggested in the literature for finding the optimal solution of the resource-relaxed problem quickly [18], [20]. However, in branch and bound based procedures only a few new arcs are introduced when generating child nodes, and we can use the optimal solution of the parent node to initialize the search in a child node. This idea occurs e.g., in [14] for speeding up the steepest ascent method applied to the net present value problem. We will show how to recompute the optimal solutions using the primal-dual maximum cost flow algorithm. On the other hand, when an upper bound on the optimum has been found, we also wish to compute tight time windows for activities, that is, the earliest and latest time points beyond which it is no use to execute the activity because the objective function is guaranteed to be greater than the known upper bound.

3.1 Updating the Primal and Dual Optimal Solutions

Branch and bound algorithms explore a tree defined by a branching scheme (cf. Section 4). In each node visited by the algorithm a lower bound is computed with respect to the constraints of the original problem and additional constraints determined during the search. Let $N(E, \delta) = (V^+, A^+ \cup E, \delta)$ be the network associated with a tree-node q, where E contains those arcs added by the branch and bound algorithm along the path leading to the node (Section 4), and $\delta : A^+ \to \mathbb{Z}$ defines the arc lengths. We generate child nodes by adding new temporal constraints to the network, that is, we obtain each child node by adding a set of arcs E' along with their lengths to $N(E, \delta)$. Since we know the lower bound in node q, the optimal primal, X^*, and dual, S^*, solutions are available and can be used to speed up the computation of the lower bound in each child node. Namely, if E' is the set of arcs added to $N(E, \delta)$ with lengths δ', first we update S^* by these arcs to obtain a dual feasible solution S for $N(E'', \delta'')$, where $E'' = E \cup E'$ and $\delta''_{i,j} := \delta_{i,j}$ for each $(i, j) \in (E \cup A^+) \setminus E'$, $\delta''_{i,j} := \max\{\delta_{i,j}, \delta'_{i,j}\}$ for each $(i, j) \in E' \cap (E \cup A^+)$ and $\delta''_{i,j} := \delta'_{i,j}$ for each $(i, j) \in E' \setminus (E \cup A^+)$. To this end, let $S := S^*$ and we check each arc $(i, j) \in A^+ \cup E \cup E'$ wether $S_j - S_i \geq \delta''_{i,j}$. If not, then update $S_j := S_i + \delta''_{i,j}$. We repeat this procedure until no more changes occur or a positive cycle is found in which case the dual problem has no feasible solution. For details, see [1].

With a feasible dual solution, we can update the primal solution to ensure that the complementary slackness conditions are satisfied. Namely, if $rc_{i,j}(S) < 0$ then let $X_{i,j} = 0$; if $rc_{i,j}(S) > 0$ then let $X_{i,j} = u_{i,j}$; while if $rc_{i,j}(S) = 0$, $X_{i,j} = X^*_{i,j}$. Clearly, X can be computed in linear time in the size of $N(E'', \delta'')$.

Finally, we can apply any variant of the primal-dual maximum cost flow algorithm to X and S to obtain a pair of optimal solutions for $N(E'', \delta'')$.

In practice this method is very efficient as E' usually contains only a few arcs and therefore only a small fraction of the distances $S_i - S_j$, $(i, j) \in A^+ \cup E \cup E'$ change. Therefore, X and X^* usually differ only on a few arcs (cf. definition of $rc_{i,j}(S)$).

3.2 Tightening the Domains of Variables

Suppose the algorithm for solving RCPSPWET has found one ore more feasible solutions (respecting constraints (i) and (ii)). Therefore, a finite upper bound UB is available on the value of the optimal solution. The *domain* of activity $j \in V$ in node q of the search tree is the time interval $[\delta_{0,j}(q), -\delta_{j,0}(q)]$ in which S_j may take a value (see the next section). However, if, say, $t_j > 0$, then possibly $\delta_{0,j}(q) < S_j$ or $S_j < -\delta_{j,0}(q)$ in any feasible solution with objective function value smaller than UB and reachable from node q. We propose a simple procedure for tightening the domains of activities using only the temporal constraints and the upper bound UB.

Consider the network $N(E, \delta) = (V^+, A^+ \cup E, \delta)$ associated with a node of the branch and bound tree. For each activity j, define the function

$$f_j(s) = \min\{v(S) \mid S_j = s \text{ and } S \text{ is dual feasible for } N(E, \delta)\}.$$

Proposition 1. *$f_j(s)$ is convex on the interval* $[\delta_{0,j}, -\delta_{j,0}]$.

With these functions, define the quantities

$$S_j^{\min} = \min\{s \in \mathbb{Z} \mid s \geq \delta_{0,j} \text{ and } f_j(s) \leq UB\}$$

and

$$S_j^{\max} = \max\{s \in \mathbb{Z} \mid s \le -\delta_{j,0}, \text{ and } f_j(s) \le UB\},$$

where UB is the best known upper bound. We discuss the computation of $S_j^{\max}$ in detail, the method for $S_j^{\min}$ being similar. After the computations we can set $\delta_{0,j} = S_j^{\min}$ and $\delta_{j,0} = -S_j^{\max}$.

Let X^* and S^* constitute a pair of primal and dual optimal solutions for $N(E,\delta)$. We may assume that $-\delta_{j,0} \in [\delta_{0,j}, \infty]$, otherwise the dual problem is infeasible. If the common primal and dual objective function value is greater than or equal to UB, then the dual feasible solutions of the maximum cost flow problem with respect to $N(E,\delta)$ cannot contain a better solution than the one with cost UB. In this case there is no use to compute time windows. So assume that the optimal objective function value is smaller than UB.

Algorithm 1. Computation of $S_j^{\max}$ (input: $N(E,\delta)$, UB; output: $S_j^{\max}$)

```
1: Let $\delta^0_{0,j} = -\delta_{j,0}$, $X^0_{0,j} = \infty$, $\delta^0_{i,j} = \delta_{i,j}$ and $X^0_{i,j} = X^*_{i,j}$ for $(i,j) \in A^+ \setminus \{(0,j)\}$.
2: Apply the Primal-Dual Maximum Cost Flow algorithm (sketched below) to $N(E,\delta)$
   starting from $X^0$, and $S^*$. Stop the algorithm when either the solution becomes
   primal feasible or the dual objective function value becomes greater than $UB$ after
   the change of the dual variables. Upon termination, let $\bar{X}$ and $\bar{S}$ denote the values
   of the primal and the dual variables, respectively.
3: Let $S'$ denote the value of the dual variables before the last increase. If $v(\bar{S}) - UB > 0$,
   $\lambda = (v(\bar{S}) - UB)/(v(\bar{S}) - v(S'))$, otherwise let $\lambda = 0$. Then,
   $S_j^{\max} := \lfloor \lambda S'_j + (1-\lambda)\bar{S}_j \rfloor$.
```

Algorithm 1 for computing $S_j^{\max}$ consists of three main steps. In the first step $\delta_{0,j}$ is increased to $-\delta_{j,0}$. With this modified $\delta_{0,j}$ value, $rc_{0,j}(S^*) > 0$ unless $S^*_j = -\delta_{0,j}$ in which case there is nothing to compute. To maintain the complementary slackness optimality conditions, $X^0_{0,j}$ is set to its upper bound. This way 0 becomes a sink node and j becomes a source node. All other nodes have zero excess.

The *Primal-Dual Maximum Cost Flow algorithm* applied in the second step of the algorithm alternates between sending flow from a source node to a sink node, and increasing the value of a subset $W \subset V^+$ of dual variables taking into account the reduced costs. Namely, it tries to send flow from a source node to a sink node through directed paths in the residual network using only arcs with zero reduced costs. If no more flow can be sent from a source to a sink node, then there is a set of nodes W reachable from a source node through these paths, but W does not contain a sink node. In our case the source node is always j and the sink node is always 0. Then the dual variables corresponding to the nodes in W are increased by the same amount ε such that complementary slackness is maintained. Namely, $\varepsilon = \min\{\varepsilon_1, \varepsilon_2\}$, where $\varepsilon_1 = \min\{rc_{i,j}(S) \mid (i,j) \in A^+, i \in V^+ \setminus W, j \in W, rc_{i,j}(S) > 0\}$ and $\varepsilon_2 = \min\{-rc_{i,j}(S) \mid (i,j) \in A^+, i \in W, j \in V^+ \setminus W, rc_{i,j}(S) < 0\}$.

In the third step, if $v(\bar{S}) \le UB$, then $S_j^{\max} = \bar{S}_j$. Otherwise, when $v(S') < UB < v(\bar{S})$, the algorithm computes the point between S'_j and $\bar{S}_j$ such that the dual objective reaches UB. To make this more precise, first notice that the choice of λ ensures $\lambda v(S') + (1-\lambda)v(\bar{S}) = UB$. Moreover, we have:

Lemma 1. *The dual objective function value satisfies* $v(\lambda S' + (1-\lambda)\overline{S}) = \lambda v(S') + (1-\lambda)v(\bar{S})$.

Proof. If $v(\bar{S}) \leq UB$ then $\lambda = 0$ and we are done. So assume $v(\bar{S}) > UB$. Let W be the set of nodes whose dual variable has been changed just before the termination of the algorithm. Then node $n+1 \notin W$. This can be seen by induction as we start out from an optimal solution S^* in which $rc_{n+1,0}(S^*) = 0$. Therefore, if $n{+}1$ was reachable from node j on a path consisting of arcs with zero reduced costs, node 0 would be reachable as well. Now, since $n{+}1 \notin W$, the choice of ε and the definition of the $rc_{n+1,i}(S)$ implies that $\varepsilon \leq \min\{d_i - p_i - S_i \mid i \in W \text{ and } S_i < d_i - p_i\}$. Consequently, for $i \in W$ with $S_i \geq d_i - p_i$, increasing S_i by ε increases the dual objective function value by εt_i, and for $i \in W$ with $S_i < d_i - p_i$, increasing S_i by ε decreases the dual objective function value by εe_i. Therefore, the dual objective changes linearly when increasing the value of a subset W of dual variables by the same amount of ε. Consequently, the objective function value changes linearly during the last increase of the dual variables and the statement follows. □

We still have to show the following:

Lemma 2. $v(\lambda S' + (1-\lambda)\overline{S})$ *equals the smallest dual objective function value over all dual feasible solutions* $\tilde{S}$ *with* $\tilde{S}_j \geq \lambda S'_j + (1-\lambda)\overline{S}_j$.

Proof. Since the complementary slackness conditions are maintained by the primal-dual maximum cost flow algorithm, if we let $S = \lambda S' + (1-\lambda)\overline{S}$, $\delta_{0,j} = S_j$, $X_{0,j} = \sum_{(j,i)\in A^+} \bar{X}_{j,i} - \sum_{(i,j)\in A^+\setminus\{(0,j)\}} \bar{X}_{i,j}$, and $X_a = \bar{X}_a$ for $a \in A^+\setminus\{(0,j)\}$, X and S will be primal and dual feasible with respect to $N(E,\delta)$, and satisfy the complementary slackness conditions. Therefore, X and S constitute a pair of primal and dual optimal solutions with value $v(\lambda S' + (1-\lambda)\overline{S})$ and $S_j = \lambda S'_j + (1-\lambda)\overline{S}_j$. □

If $\lambda S'_j + (1-\lambda)\overline{S}_j$ is fractional, then clearly, we can round it down to obtain $S_j^{\max}$.

The algorithm performs at most $n + |A^+|$ dual changes, and between two changes it solves a maximum flow problem with successive augmenting paths (which is theoretically not efficient). Clearly, the maximum flow problem may be solved by any polynomial time algorithm and thus the entire algorithm may be implemented to run in polynomial time. However, in practice, our method is quite efficient as there are only a few flow augmentations between any two changes of dual variables. We illustrate all this by means of a small example:

Example 1. A simple AoN-network is depicted in Figure 1. For the sake of simplicity we suppose that i) each arc is an end-to-start precedence relation ($\delta_{i,j} = p_i$) and ii) the earliness and tardiness costs are equal for each activity ($e_j = t_j$). The optimal solution (of the resource-relaxed problem) is

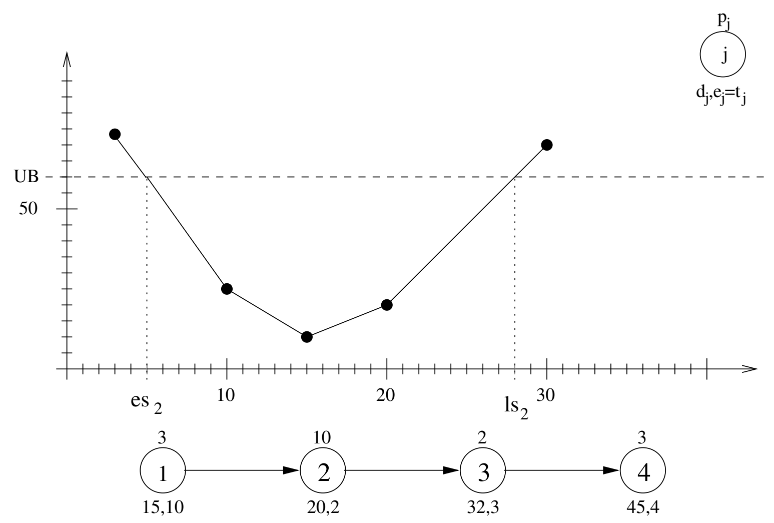

Fig. 1. Illustration of computing $S_j^{\max}$

$S^* =< 12, 15, 30, 42 >$ with cost $c = 10$. Assume that we have an upper bound $UB = 60$.

To compute $S_2^{\max}$, we start to shift activity 2 to the right. We can shift it until $S_2 = 20$, where arc $(2, 3)$ becomes tight, thus $\varepsilon = 5$. The new schedule is $S^1 =< 12, 20, 30, 42 >$, the cost is $c = 30$. Since this is smaller than the upper bound, we continue with shifting activities $\{2, 3\}$, until arc $(3, 4)$ becomes tight ($S_3 = 40$). $\varepsilon = 8$, the schedule is $S^2 =< 12, 30, 40, 42 >$ with cost $c = 70$. $c > UB$, so we can calculate $S_2^{\max}$ from S_2^1 and S_2^2. Regarding that the per-unit tardiness cost of activities $\{2, 3\}$ together is $t_2 + t_3 = 5$, we get that $S_2^{\max} = 28$, and the corresponding schedule is $< 12, 28, 38, 42 >$ with cost $c = 60$.

4 Exact Algorithm Based on Branch and Bound

A subset C of activities are in conflict, or C is a *conflicting set*, if there exists $k \in R$ with $\sum_{i \in C} r_{i,k} > b_k$. Two activities, i and j, are *resource-disjunct* if $\{i, j\}$ is a conflicting set. If $\mathcal{A}(S, t)$ is a conflicting set, then the inclusion-wise minimal subsets B of $\mathcal{A}(S, t)$ such that $\mathcal{A}(S, t) \setminus B$ is non-conflicting are called *minimal delaying alternatives* [5]. Algorithm 2 is an extension of Algorithm 3.5.1 of [15] for project scheduling with general precedence relations and non-regular objective functions (see also [6] and [14]). The extension consists of the application of domain filtering techniques in lines (15)-(18).

In Algorithm 2, L is a list of nodes and S^*_{best} is the best resource feasible solution found so far. At the beginning, L contains only the root node and S^*_{best} is empty. The search proceeds in a depth-first manner by always choosing the last node q appended to the end of L. Each node is processed only once. First

Algorithm 2. Branch-and-Bound (input: UB, output: S^*_{best})

```
1:  $L := \{r\}$, $S^*_{best} := \emptyset$
2:  Compute all-pairs-longest-paths $\delta(r)$
3:  while $L \neq \emptyset$ do
4:    Remove the last node $q$ from $L$
5:    if $S^*(q)$ is resource feasible then
6:      if $v(S^*(q)) < UB$ then
7:        $UB := v(S^*(q))$, $S^*_{best} := S^*(q)$
8:      end if
9:    else
10:     Determine the earliest time point $t$ when $\mathcal{A}(S^*(q), t)$ is conflicting
11:     $K := \emptyset$
12:     for each minimal delaying alternative $B \subset \mathcal{A}(S^*(q), t)$ do
13:       for all $i \in \mathcal{A}(S^*(q), t) \setminus B$ do
14:         Generate node $q'$ from node $q$ with additional temporal constraints $(i, j)$
            with $\delta_{i,j}(q') = p_i$ for each $j \in B$
15:         if $UB < \infty$ then
16:           Tighten the variable domains with Network Computations
17:           Tighten the variable domains with Unit-Interval Tests and Edge-
              Finding
18:         end if
19:         Compute all-pairs-longest-paths $\delta(q')$
20:         Perform Immediate-Selection for resource-disjunct activities
21:         Re-compute $S^*(q')$
22:         if node $q'$ is infeasible or dominated, or $v(S^*(q')) \geq UB$ then
23:           Delete node $q'$
24:         else
25:           $K := K \cup \{q'\}$
26:         end if
27:       end for
28:     end for
29:     Append the nodes $q' \in K$ to $L$ in non-increasing order of $v(S^*(q'))$
30:   end if
31: end while
```

it is checked whether the solution associated with the node, $S^*(q)$, is resource feasible. If it is, the node will be fathomed. But before, it is checked whether $S^*(q)$ represents a better solution than S^*_{best} (line 6). If $S^*(q)$ is not resource feasible, then the algorithm finds the first time point t when $\mathcal{A}(S^*(q), t)$ is conflicting. Then for each minimal delaying alternative with respect to $\mathcal{A}(S^*(q), t)$, a new node q' is generated (line 14) [8], [6]. If there is a feasible solution, i.e., $UB < \infty$, we tighten the time windows of activities by the network computations described in Section 3.2 and also by two well-known domain filtering techniques: the Unit-Interval Tests and Edge-Finding (see [16], [2] and [7]), lines (15)-(18). All these domain filtering techniques are invoked only once for each activity, and not until a fixpoint is reached. We have found by experimentation that it does not pay off repeating these computations until no more changes occur. Then we compute the longest paths between all pairs of activities in $V \cup \{0\}$ and store it in the $(n+1) \times (n+1)$ matrix $\delta(q')$. Using $\delta(q')$, we perform immediate selections between pairs of resource-disjunct activities. To this end, we apply the techniques of [15], pages 53-57 (see also [14]) which extend those of [4] and [6]. Notice that in the end of the computations, $\delta_{0,i}(q')$ equals the earliest start time of activity i, while $-\delta_{i,0}(q')$ is its latest start time. The node is *infeasible* if $\delta_{0,i}(q') + \delta_{i,0}(q') > 0$, or $\delta_{i,i}(q') > 0$ hold for any $i \in V$. The child node is dropped if it is infeasible, or dominated by some node already explored in the tree (we apply the subset dominance rule of [14] which is a strengthening of that proposed in [6]), or its lower bound is not smaller than the current upper bound (line 23). Those child nodes that are not deleted get appended to L in non-increasing order of their lower bounds (line 29).

We apply Algorithm 2 in two phases. In the first phase, when no feasible solution is known, we invoke the algorithm with $UB = \infty$ and stop it immediately when it encounters the first resource and time feasible solution, or proves that the problem is infeasible. If a feasible solution is found, we invoke Algorithm 2 the second time with UB equal to the value of the known feasible solution. Notice that the domain filtering techniques in lines (15)-(18) become active only during the second invocation.

5 Computational Results

To evaluate the performance of our algorithm, we used two groups of RCPSP/ max test instances, UBO20 and UBO50 generated by ProGen/max [17]. Since these test instances were generated for the makespen objective, we complemented each of them by due-date and per unit earliness and tardiness cost information. Namely, for an instance with n activities we generated n random numbers chosen uniformly from the interval $[1, 1.5C_{\max}]$, where $C_{\max}$ is the makespan of the resource-relaxed problem. Then we sorted these numbers in ascending order and assigned the i-th member of the sorted list to the i-th activity of the instance. The per unit earliness and tardiness costs were chosen uniformly at random from the interval $[1, 100]$. We implemented the complete algorithm in C++ . We run the tests on a PC with 2.4 GHz Xeon CPU, using Windows 2000 operating system. To make a fair comparison with the method of Schwindt [18], we also solved all instances with his program which he kindly provided to us. In all of the following tables, column "TW" represents our algorithm.

5.1 Results on UBO20 Instances

All of these instances consists of 20 activities. We run both algorithms with a 100-second time limit. Out of the 90 instances, 20 were infeasible. As we can see in Table 1, our solver (TW) was capable to solve all the 70 feasible instance within 22 seconds at most, while the code of Schwindt exceeded the 100-second time limit in three cases.

Having no access to the number of search-tree nodes generated by the solver of Schwindt, we do not provide a comparison of the sizes of search-trees.

Table 1. Results on UBO20 instances

	TW	Schwindt
# inst. solved to opt. in 100s	70	67
max solution time	21.97s	$> 100s^a$
avg. solution time[b]	0.22s	1.32s

[a] There were 3 instances which took more than 100s.
[b] On instances both solver solved to optimality.

5.2 Results on UBO50 Instances

Each of the 90 instances in this group consists of 50 activities, and we performed a more thorough evaluation. We run the tests with 100, 200, 400-second time limits. Out of the 90 instances, there were 17 instances which did not admit a feasible solution. Both algorithms proved infeasibility of these instances within a few seconds. For all other instances our algorithm found a feasible solution already within 100 seconds. However, the method of Schwindt did not find a feasible solution for 3 additional instances within 100s, and for 1 additional instance within 200s, while it found a feasible solution for all feasible instances within 400s, see Table 2.

Table 2. Number of feasible UBO50 instances for which no feasible solution was found

Time limit	TW	Schwindt
100s	0	3
200s	0	1
400s	0	0

Table 3 shows the number of instances solved to optimality by the algorithms. Our solver was able to solve more instances to optimality with respect to all three time limits. Moreover, as the time limit increased, our solver solved more new instances to optimality than that of Schwindt.

Table 3. Number of UBO50 instances solved to optimality.

Time limit	TW	Schwindt	Both
100s	32	24	24
200s	35	26	26
400s	39	28	28

Table 4 depicts the average running times on those instances solved to optimality by both solvers within the respective time limits. As can be seen, our solver is faster on average.

Table 4. Average solution times on UBO50 instances

Time limit	TW	Schwindt
$100s$	$8.64s$	$22.38s$
$200s$	$11.89s$	$33.01s$
$400s$	$13.32s$	$53.19s$

Finally, Table 5 compares the algorithms on those instances not solved to optimality by either of two solvers. The columns "TW", "Equal", and "Schwindt" provide the number of instances (within this set of instances) on which our algorithm found a better solution, the two solvers found solutions of equal total tardiness, and the program of Schwindt found a better solution, respectively, within the given time limits. Our method clearly performs better than that of Schwindt in this respect, too.

Table 5. UBO50: Number of better non-optimal solutions found

	TW	Equal	Schwindt
$100s$	22	13	6
$200s$	21	12	5
$400s$	21	8	5

The above results show that our solver outperforms that of Schwindt in all aspects examined.

6 Conclusions

In this paper we have described a simple method for computing tight time windows for activities when solving the RCPSPWET problem which enables us to apply many domain filtering techniques known in the Constraint-based Scheduling community. Our preliminary computational results indicate that it is worthwhile to make these computations as our method clearly outperforms the only method published in the literature.

A major question is how to speed up our method. We have observed that in 60%-70% of the cases our method is able to reduce the domains of activities and this is independent from the depth of the node in the search-tree. Nevertheless, it would be interesting to find out in advance for which activities to perform the computations in a search-tree node.

Acknowledgments. The authors are grateful to the three anonymous referees for helpful comments that helped to improve the presentation.

References

1. R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
2. Ph. Baptiste, C. Le Pape, W.P.M. Nuijten, *Constraint-Based Scheduling. Applying Constraint Programming to Scheduling Problems*, Kluwer Academic Publishers, Boston, 2001.
3. J.C. Beck, P. Refalo, A hybrid approach to scheduling with earliness and tardiness costs, *Annals of Operations Research*, 118 (2003) 49-71.
4. P. Brucker, S. Knust, A. Schoo, O. Thiele, A branch and bound algorithm for the resource-constrained project scheduling problem, *Eur. J. Oper. Res.* 107 (1998) 272-288.
5. E.L. Demeulemeester, W.S. Herroelen, A branch and bound procedure for the multiple resource constrained project scheduling problem, *Management Sci.* 38 (1992) 1803-1818.
6. B. De Reyck, W.S. Herroelen, A branch and bound procedure for the resource-constrained project scheduling problem with generalized precedence relations, *Eur. J. Oper. Res.* 111 (1998) 152-174.
7. U. Dorndorf, *Project scheduling with time windows: From theory to applications*, Physica-Verlag, 2002.
8. O. Icmeli, S. Erengüç, A branch and bound procedure for the resource-constrained project scheduling problem with discounted cash flows, *Management Sci.* 42 (1996) 1395-1408.
9. A. Kéri, T. Kis, Primal-dual combined with constraint propagation for solving RCPSPWET. In: G. Kendall, L. Lei and M. Pinedo (eds.), *Proc. of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications*, New York University, New York, July, 2005, pp. 748-751 (electronic edition).
10. A. Kéri, T. Kis, Primal-Dual combined with constraint propagation for solving RCPSPWET, In: H-D. Haasis, H. Kopfer, J. Schönberger (eds.), *Operation Research Proceedings*, September 7-9, 2005, Bremen, Germany, Springer, pp. 685-690.
11. L. Khatib, P. Morris, R. Morris, F. Rossi, Temporal Constraint Reasoning with Preferences, *Proc. Seventieth Int. Joint Conf. Artif. Intell. (IJCAI'01)*, August 4-10, 2001, Seattle, USA, pp. 322-327.
12. T.K.S. Kumar, Fast (incremental) algorithms for useful classes of simple temporal problems with preferences, *Proc. Twentieth Int. Joint Conf. Artif. Intell. (IJCAI'07)*, January 6-12, 2007, Hyderabad, India, pp. 1954-1959.
13. P. Morris, R. Morris, L Khatib, S. Ramakrishnan, A. Bachmann, Strategies for global optimization of temporal preferences, *LNCS 3258: Proc. Tenth Int. Conf. Princip. Practice Artif. Intell. (CP 2004)*, Toronto, Canada, September 27-October 1, 2004, pp. 408-422.
14. K. Neumann, J. Zimmermann, Exact and truncated branch and bound procedures for resource-constrained project scheduling with discounted cash flows and general temporal constraints, *Central Eur. J. of Oper. Res.*, 10:4 (2002) 357-380.
15. K. Neumann, C. Schwindt, J. Zimmermann, *Project Scheduling with Time Windows and Scarce Resources*, 2nd ed. Springer, Berlin, 2003.
16. W.P.M. Nuijten, *Time and resource constrained scheduling: A constraint satisfaction approach*, PhD Thesis, Eindhoven University of Technology, 1994.

17. C. Schwindt, ProGen/max: A New Problem Generator for Different Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags, *Technical report* WIOR-449, University of Karlsruhe, Karlsruhe, 1996. (URL: http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html)
18. C. Schwindt, Minimizing earliness-tardiness costs of resource constrained projects, In: K. Inderfurth, G. Schwödiauer, W. Domschke, F. Juhnke, P. Kleinschmidt, G. Wäscher (eds.), *Operations Research Proceedings* 1999. Springer, Berlin, 2000, pp. 402-407.
19. C. Schwindt, J. Zimmermann, A steepest ascent approach to maximizing the net present value of projects, *Math. Methods of Oper. Res.* 53 (2001) 435-450.
20. M. Vanhoucke, E.L. Demeulemeester, W.S. Herroelen, An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem, *Annals of Oper. Res.* 102 (2001) 179-196.
21. M. Wennink, *Algorithmic support for automated planning boards*, PhD Thesis, Eindhoven University of Technology, 1995.