KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 9907

AN EXACT PROCEDURE FOR THE UNCONSTRAINED WEIGHTED EARLINESS-TARDINESS PROJECT SCHEDULING PROBLEM

by
M. VANHOUCKE
E. DEMEULEMEESTER
W. HERROELEN

D/1999/2376/07

AN EXACT PROCEDURE FOR THE UNCONSTRAINED WEIGHTED EARLINESS-TARDINESS PROJECT SCHEDULING PROBLEM

Mario VANHOUCKE • Erik DEMEULEMEESTER • Willy HERROELEN

January 1999

Operations Management Group
Department of Applied Economics
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven (Belgium)
Phones: 32-16-32 69 65, 32-16-32 69 72, 32-16-32 69 70, Fax 32-16-32 67 32
E-mail: <first name>.<name>@econ.kuleuven.ac.be

AN EXACT PROCEDURE FOR THE UNCONSTRAINED WEIGHTED EARLINESS-TARDINESS PROJECT SCHEDULING PROBLEM

Mario VANHOUCKE • Erik DEMEULEMEESTER • Willy HERROELEN

ABSTRACT

In this paper we study the unconstrained project scheduling problem with weighted earliness-tardiness penalty costs subject to zero-lag finish-start precedence constraints. Each activity of this unconstrained project scheduling problem has a known deterministic due date, a unit earliness penalty cost and a unit tardiness penalty cost. The objective is to schedule the activities in order to minimize the weighted earliness-tardiness penalty cost of the project, in the absence of constraints on the use of resources. With these features the problem setting becomes highly attractive in just-in-time environments.

We introduce a two-step recursive algorithm. The first step consists of a forward pass procedure which schedules the activities such that they finish at their due date or later. The second step applies a recursive search in which the activities are eventually shifted backwards (towards time zero) in order to minimize the weighted earliness-tardiness cost of the project. The procedure has been coded in Visual C++, version 4.0 under Windows NT 4.0 and has been validated on a randomly generated data set.

Keywords: *Project Scheduling; Weighted earliness-tardiness costs; Optimal search*

1. Introduction

Most of the work in project scheduling has focused on regular measures of performance. A regular measure of performance is a nondecreasing function of the activity completion times (in the case of a minimization problem), with the minimization of the project duration as the most popular one. Other examples are the minimization of the mean flowtime, the mean lateness, the mean tardiness and the percentage of jobs tardy.

In recent years scheduling problems with nonregular measures of performance have gained increasing attention. A nonregular measure of performance is a measure for which the above definition does not hold. A popular nonregular measure of performance in the literature is the maximization of the net present value (*npv*) of the project. In this case, a positive or negative cash flow is assigned to each activity and the objective is to schedule the activities in order to maximize the total net present value of the project. We can distinguish between procedures for the unconstrained max-*npv* project scheduling problem and those for the resource-constrained max-*npv* project scheduling problem. For an overview of the literature, we refer to Herroelen et al. (1997) and De Reyck and Herroelen (1998).

Another nonregular measure of performance, which is gaining attention in JIT environments, is the minimization of the weighted earliness-tardiness penalty costs of the activities in a project. In this problem, a due date, a unit earliness penalty cost and a unit tardiness penalty cost are assigned to the activities and the objective is to schedule the activities to minimize the weighted penalty cost of the project. This problem often occurs in practice since many project schedulers have to deal with due dates and penalty costs. Costs of earliness include extra storage requirements and idle times and implicitly incur opportunity costs. Tardiness leads to customer complaints, loss of reputation and profits, monetary penalties or goodwill damages. The problem is faced by many firms hiring subcontractors, maintenance crews as well as research teams. Again, a distinction can be made between the unconstrained weighted earliness-tardiness project scheduling problem (denoted as $cpm|early/tardy$, according to the classification scheme of Herroelen et al. (1998)), where activities are only subject to precedence constraints and no constraints are imposed on the use of resources, and the constrained weighted earliness-tardiness project scheduling problem where the activities are also subject to renewable resource constraints ($m,1|cpm|early/tardy$).

In this paper we present an exact algorithm for solving problem $cpm|early/tardy$ (further denoted as *WETPSP*, i.e. the **w**eighted **e**arliness-**t**ardiness **p**roject **s**cheduling **p**roblem). To the best of our knowledge, no exact algorithm has yet been suggested for the *WETPSP*. The proposed methodology exploits the basic idea that the earliness-tardiness costs of a project can be minimized by first scheduling activities at their due date or at a later time instant if forced so by binding precedence constraints, followed by a recursive search which computes the optimal displacement for those activities for which a shift towards time zero proves to be beneficial. The organisation of the paper is as follows. In section 2 we give a problem formulation. Section 3 describes an exact solution procedure while section 4 is reserved for an illustration by means of a numerical example. In section 5 we report extensive computational results on a benchmark problem set. Section 6 contains overall conclusions and suggestions for future research.

2. The deterministic unconstrained weighted earliness-tardiness project scheduling problem (*WETPSP*)

The *WETPSP* involves the scheduling of project activities in order to minimize the weighted earliness-tardiness penalty costs in the absence of resource constraints. The project is represented by an activity-on-the-node (AON) network $G=(N,A)$ where the set of nodes, N, represents activities and the set of arcs, A, represents finish-start precedence constraints with a time-lag of zero. The activities are numbered from the dummy start activity 1 to the dummy end activity n, such that $j>i$ for each arc (i,j). The duration of an activity is denoted by d_i $(1 \leq i \leq n)$ and its known deterministic due date by h_i. The completion time of activity i is denoted by the nonnegative integer variable f_i $(1 \leq i \leq n)$. The earliness of activity i can be computed as $E_i = \max(0, h_i - f_i)$ and its tardiness as $T_i = \max(0, f_i - h_i)$. If e_i and t_i respectively denote the per unit earliness and tardiness penalty cost of activity i, its total earliness-tardiness cost is $e_iE_i + t_iT_i$. In the sequel we assume, without loss of generality, that $h_1 = 0$ and $h_n = \infty$ while $e_1 = t_1 = \infty$ and $e_n = t_n = 0$. The *WETPSP* can be formulated as follows:

$$\text{Minimize} \sum_{i=2}^{n-1} (e_i E_i + t_i T_i) \qquad [1]$$

Subject to

$$f_i \leq f_j - d_j \qquad \forall (i,j) \in E \qquad [2]$$

$$E_i \geq h_i - f_i \qquad \forall i \in N \qquad [3]$$

$$T_i \geq f_i - h_i \qquad \forall i \in N \qquad [4]$$

$$f_1 = 0 \qquad [5]$$

The objective in Eq. 1 minimizes the weighted earliness-tardiness cost of the project. The constraint set given in Eq. 2 maintains the finish-start precedence relations among the activities. Eq. 3 and Eq. 4 compute the earliness and tardiness of each activity and Eq. 5 forces the dummy start activity to end at time zero.

In the next section we describe an exact recursive search procedure for the *WETPSP* as formulated above.

3. The exact solution procedure

3.1 Description

The proposed recursive algorithm consists of two steps. Step 1 determines the so-called *due date tree*, *DT*, using a forward pass procedure. The forward procedure forces the finishing time f_j of each activity j to be greater than or equal to its due date h_j. Upon terminating step 1 each node in the due date tree, except the dummy end activity n, has at most one incoming arc.

In step 2 the due date tree is the subject of a recursive search (starting from the dummy end activity n) in order to identify sets of activities (*SA*) that might be shifted backwards in time (towards time zero) in order to decrease the weighted earliness-tardiness cost of the project. Due to the structure of the recursive search it can never happen that a forward shift of a set of activities (away from time zero) can lead to a decrease of the weighted earliness-tardiness cost. In fact, all the activities are scheduled in

step 1 at their due date or later, therefore it can never be advantageous to increase the completion times of these activities.

When a set of activities *SA* is found for which a backward shift leads to a reduction in the earliness-tardiness cost, the algorithm computes its minimal displacement interval and updates the due date tree *DT* as follows. The arc (i,j) which connects a node $i \in SA$ to a node $j \notin SA$ in the due date tree *DT* is removed from it. The minimal displacement interval of the set of activities *SA* under consideration is computed as follows. Compute $v_{k^*l^*} = \min_{\substack{(k,l)\in A \\ k\notin SA \\ l\in SA}} \{f_l - d_l - f_k\}$ and $w = \min_{\substack{y\in SA \\ f_y > h_y}} \{f_y - h_y\}$. If $v_{k^*l^*} < w$, arc (k^*,l^*) is added to the due date tree *DT*. If the node k^* does not belong to an arc of the due date tree *DT* then arc (k^*,n) is added to the due date tree *DT*. If $v_{k^*l^*} \geq w$ and the set of activities *SA* consists of more than one activity, then arc (i,n) is added to the due date tree *DT*. In doing so, we make sure that the due date tree *DT* is never disconnected into two subtrees during the performance of the recursive search.

The completion times of the activities in the set of activities *SA* for which the displacement has been computed are decreased by the minimal displacement $\min\{v_{k^*l^*}, w\}$ and the algorithm repeats the recursive search. If no further shift can be accomplished, the algorithm stops and the completion times of the activities of the project with its corresponding weighted earliness-tardiness cost are reported.

3.2 The algorithm

When f_j denotes the finishing time of activity j, when P_j denotes the set of its immediate predecessors, when *DT* denotes the due date tree, when *SA* denotes a set of activities for which the per unit earliness-tardiness cost will be denoted by *ET* and *CA* denotes the set of already considered activities, the two steps and the recursive algorithm can be written as follows:

STEP 1. COMPUTE DUE DATE TREE
$DT = \varnothing$;
$f_1 = h_1 = 0$;
Do for $j = 2$ to n
$f_{max} = -1$;
Do $\forall i \in P_j$
If $f_i > f_{max}$ **then** $f_{max} = f_i$ and $i^* = i$;
$f_{max} = f_{max} + d_j$;
If $j < n$ **then** $f_j = \max\{h_j; f_{max}\}$ **else** $f_j = f_{max}$;
If $f_{max} > h_j$ **then** $DT = DT \cup (i^*,j)$;
Do for $j = 1$ to n-1
If $\neg\exists(i,j) \in DT$ and $\exists(j,k) \in DT$ **then** $DT = DT \cup (j,n)$;

STEP 2.
$CA = \varnothing$;
Do *RECURSION(n)* $\rightarrow SA'$, ET' (parameters returned by the recursive function);
Report the optimal completion times of the activities and the weighted earliness-tardiness cost of the project.

RECURSION(NEWNODE)

$SA = \{newnode\}$ and $CA = CA \cup \{newnode\}$;
If $f_{\text{newnode}} > h_{newnode}$ **then** $ET = -t_{newnode}$ **else** $ET = e_{newnode}$;
Do $\forall i | i \notin CA$ and i precedes *newnode* in the due date tree DT:
RECURSION(i) $\rightarrow SA', ET'$
If $ET' \geq 0$ **then**
Set $SA = SA \cup SA'$ and $ET = ET + ET'$;
Else
$DT = DT \backslash (i, newnode)$;
Compute $v_{k^*l^*} = \min_{\substack{(k,l)\in A \\ k\notin SA' \\ l\in SA'}} \{f_l - d_l - f_k\}$ and $w = \min_{\substack{y\in SA' \\ f_y > h_y}} \{f_y - h_y\}$;
If $v_{k^*l^*} < w$ **then**
If $\neg\exists(r,k^*) \in DT$ and $\neg\exists(k^*,s) \in DT$ **then** $DT = DT \cup (k^*,n)$;
$DT = DT \cup (k^*,l^*)$;
else
If $|SA'| > 1$ **then** $DT = DT \cup (i,n)$;
Do $\forall j \in SA'$: set $f_j = f_j - \min\{v_{k^*l^*}, w\}$;
Go to STEP 2;
Do $\forall i | i \notin CA$ and i succeeds *newnode* in the due date tree DT:
RECURSION(i) $\rightarrow SA', ET'$
If $ET' < 0$ **then**
Set $SA = SA \cup SA'$ and $ET = ET + ET'$;
Else
$DT = DT \backslash (newnode, i)$;
If $|SA'| > 1$ **then** $DT = DT \cup (i,n)$;
Return;

Notice that the due date tree DT contains several subtrees, each connected with the dummy end activity n. When a particular subtree is subject to a recursive search and no displacement can be found, we make the link between that subtree and the dummy end activity n inactive. In doing so, the recursive search procedure will dominate this link and will not search for a set of activities of that particular dominated subtree. When later during the performance of the recursive search procedure, due to the displacement of a set of activities, an arc is added between an activity of the inactive subtree and another activity, the inactive subtree will be activated again by making its link with the dummy end activity n active again. The subtree can now again be subject to a recursive search and eventually a set of activities that will be shifted towards time zero can be found.

4. Numerical example

Consider the AON project network given in Figure 1. The number above each node denotes the activity duration, while the numbers below each node denote the due date and the unit penalty cost respectively. Notice that, for ease of representation, we assume the unit earliness costs to equal the unit tardiness costs. Notice also that the recursive algorithm is able to handle problems for which not all the activities have a due date. In the example, activity 5 has no due date constraint. The recursive algorithm runs as follows.

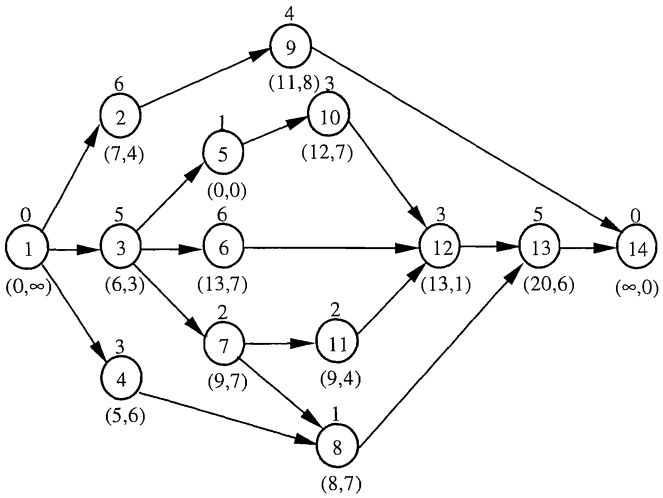

Figure 1. A project network with due dates and unit penalty costs

STEP 1. COMPUTE DUE DATE TREE

The forward pass algorithm computes the finish times of the activities as f_1=0, f_2=7, f_3=6, f_4=5, f_5=7, f_6=13, f_7=9, f_8=10, f_9=11, f_{10}=12, f_{11}=11, f_{12}=16, f_{13}=21 and f_{14}=21. The algorithm constructs the due date tree DT = {(3,5),(3,14),(6,12),(6,14),(7,8),(7,11), (7,14),(12,13)}. The due date tree consists of three subtrees which are represented in Figure 2 in bold. The links with the dummy end activity n are created during the execution of the last do-loop of step 1.

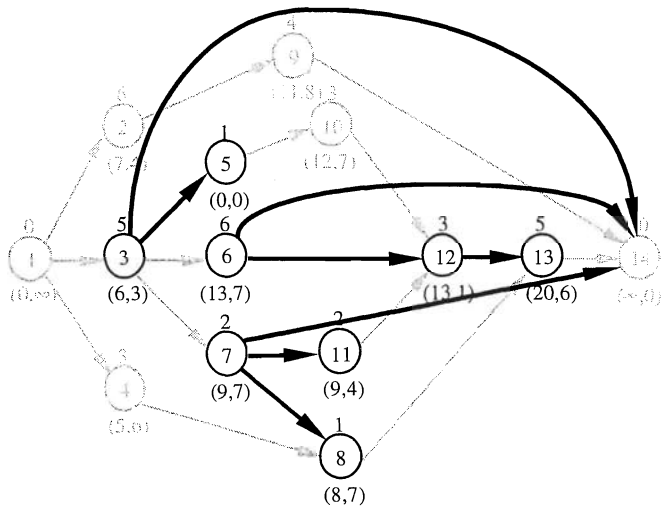

Figure 2. Due date tree generated in step 1

STEP 2. Set $CA=\varnothing$. The algorithm will now perform a recursive search starting with node 14.

RECURSION(14)

$SA = \{14\}$, $CA = \{14\}$. As $f_{14} < h_{14}$, $ET = 0$.

RECURSION(7) : predecessor node 7

$SA = \{7\}$, $CA = \{7,14\}$. As $f_7 = h_7$, $ET = 7$.

RECURSION(8) : successor node 8

$SA = \{8\}$, $CA = \{7,8,14\}$. As $f_8 > h_8$, $ET = -7$.

$ET' = -7 < 0$: $SA = \{7,8\}$, $ET = -7 + 7 = 0$.

RECURSION(11) : successor node 11

$SA = \{11\}$, $CA = \{7,8,11,14\}$. As $f_{11} > h_{11}$, $ET = -4$.

$ET' = -4 < 0$: $SA = \{7,8,11\}$, $ET = -4 + 0 = -4$.

$ET' = -4 < 0$: The set of activities $\{7,8,11\}$ must be shifted backwards towards time zero. Delete the arc (7,14) from the due date tree. Compute $v_{37} = \min\{(f_7-d_7-f_3),(f_8-d_8-f_4)\}=1$ and $w=\min\{f_8-h_8,f_{11}-h_{11}\}=2$. Since $v_{37}<w$ and $k^*=3$ belongs to arc (3,5) of the due date tree, we only add the arc (3,7) to the due date tree. Decrease the completion times of the activities in SA' with $\min\{v_{37},w\} = 1 : f_7 = 8, f_8 = 9$ and $f_{11} = 10$. Repeat STEP 2 with the updated due date tree $DT = \{(3,5),(3,7),(3,14),(6,12),(6,14),(7,8),(7,11),(12,13)\}$ shown in bold in Figure 3.

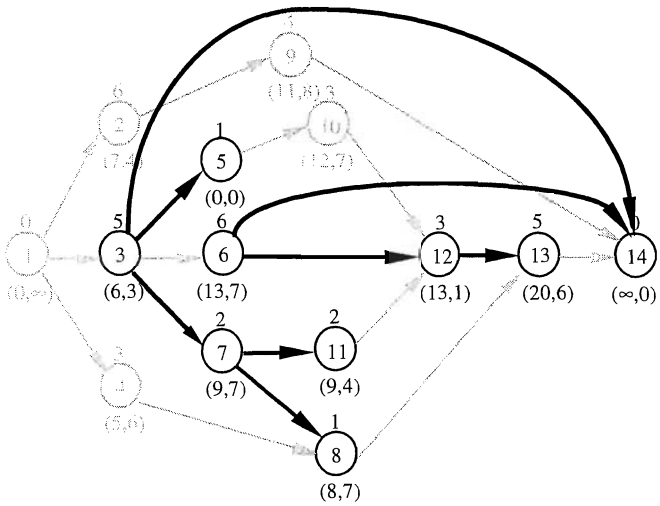

Figure 3. $DT = \{(3,5),(3,7),(3,14),(6,12),(6,14),(7,8),(7,11),(12,13)\}$

STEP 2. Set $CA=\varnothing$. The algorithm again starts a recursive search starting with node 14.

RECURSION(14)

$SA = \{14\}$, $CA = \{14\}$. As $f_{14} < h_{14}$, $ET = 0$.

RECURSION(6) : predecessor node 6

$SA = \{6\}$, $CA = \{6,14\}$. As $f_6 = h_6$, $ET = 7$.

RECURSION(12) : successor node 12

$SA = \{12\}$, $CA = \{6,12,14\}$. As $f_{12} > h_{12}$, $ET = -1$.

RECURSION(13) : successor node 13

$SA = \{13\}$, $CA = \{6,12,13,14\}$. As $f_{13} > h_{13}$, $ET = -6$.

$ET' = -6 < 0$: $SA = \{12,13\}$, $ET = -6 + (-1) = -7$.

$ET' = -7 < 0$: $SA = \{6,12,13\}$, $ET = -7 + 7 = 0$.

$ET' = 0 \geq 0$: $SA = \{6,12,13,14\}$, $ET = 0 + 0 = 0$.

No displacement has been found for the subtree: make the arc (6,14) inactive.

RECURSION(3) : predecessor node 3

$SA = \{3\}$, $CA = \{3,6,12,13,14\}$. As $f_3 = h_3$, $ET = 3$.

RECURSION(5) : successor node 5

$SA = \{5\}$, $CA = \{3,5,6,12,13,14\}$. As $f_5 > h_5$, $ET = 0$.

$ET' = 0$: delete arc (3,5) from DT as shown in Figure 4.

Since $|SA'| = 1$, no connection is made between node 5 and node 14.

RECURSION(7) : successor node 7

$SA = \{7\}$, $CA = \{3,5,6,7,12,13,14\}$. As $f_7 < h_7$, $ET = 7$.

RECURSION(8) : successor node 8

$SA = \{8\}$, $CA = \{3,5,6,7,8,12,13,14\}$. As $f_8 > h_8$, $ET = -7$.

$ET' = -7 < 0$: $SA = \{7,8\}$, $ET = 7 + (-7) = 0$.

RECURSION(11) : successor node 11

$SA = \{11\}$, $CA = \{3,5,6,7,8,11,12,13,14\}$. As $f_{11} > h_{11}$, $ET = -4$.

$ET' = -4 < 0$: $SA = \{7,8,11\}$, $ET = 0 + (-4) = -4$.

$ET' = -4 < 0$: $SA = \{3,7,8,11\}$, $ET = -4 + 3 = -1$.

$ET' = -1 < 0$: The set of activities $\{3,7,8,11\}$ must be shifted backwards towards time zero. Delete the arc (3,14) from the due date tree. Compute $v_{13} = \min\{(f_3-d_3-f_1),(f_8-d_8-f_4)\}=1$ and $w=\min\{f_8-h_8,f_{11}-h_{11}\}=1$. Since $v_{13}=w$ and $|SA'| = 4 > 1$, we add the arc (3,14) to the due date tree. Decrease the completion times of the activities in SA' with $\min\{v_{13},w\} = 1$: $f_3 = 5$, $f_7 = 7$, $f_8 = 8$ and $f_{11} = 9$. Repeat STEP 2 with the updated due date tree $DT = \{(3,7),(3,14),(6,12),(6,14),(7,8),(7,11),(12,13)\}$ shown in bold in Figure 4.

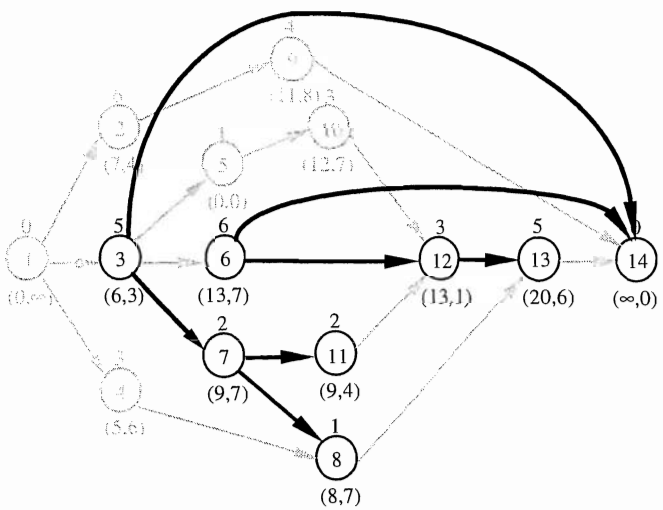

Figure 4. $DT = \{(3,7),(3,14),(6,12),(6,14),(7,8),(7,11),(12,13)\}$

STEP 2. Set $CA=\varnothing$. The algorithm again starts a recursive search starting with node 14.

RECURSION(14)

$SA = \{14\}$, $CA = \{14\}$. As $f_{14} < h_{14}$, $ET = 0$.

Arc (6,14) is inactive.

RECURSION(3) : predecessor node 3

$SA = \{3\}$, $CA = \{3,14\}$. As $f_3 < h_3$, $ET = 3$.

RECURSION(7) : successor node 7

$SA = \{7\}$, $CA = \{3,7,14\}$. As $f_7 < h_7$, $ET = 7$.

RECURSION(8) : successor node 8

$SA = \{8\}$, $CA = \{3,7,8,14\}$. As $f_8 = h_8$, $ET = 7$.

$ET' = 7 > 0$: delete arc (7,8) from DT.

Since $|SA'| = 1$, no connection is made between node 8 and node 14.

RECURSION(11) : successor node 11

$SA = \{11\}$, $CA = \{3,7,8,11,14\}$. As $f_{11} = h_{11}$, $ET = 4$.

$ET' = 4 > 0$: delete arc (7,11) from DT.

Since $|SA'| = 1$, no connection is made between node 11 and node 14.

$ET' = 7 > 0$: delete arc (3,7) from DT.

Since $|SA'| = 1$, no connection is made between node 7 and node 14.

$ET' = 3 \geq 0$: $SA = \{3,14\}$, $ET = 3 + 0 = 3$.

RETURN;

No set of activities can be shifted towards time zero to decrease the weighted earliness-tardiness cost of the due date tree and there are no active links to the dummy end activity n left, so the algorithm stops. The due date tree $DT = \{(3,14),(6,12),(6,14),(12,13)\}$ is given in bold in Figure 5. The completion times of the activities are $f_1=0$, $f_2=7$, $f_3=5$, $f_4=5$, $f_5=7$, $f_6=13$, $f_7=7$, $f_8=8$, $f_9=11$, $f_{10}=12$, $f_{11}=9$, $f_{12}=16$, $f_{13}=21$, and $f_{14}=21$. The weighted earliness-tardiness cost amounts to 26.

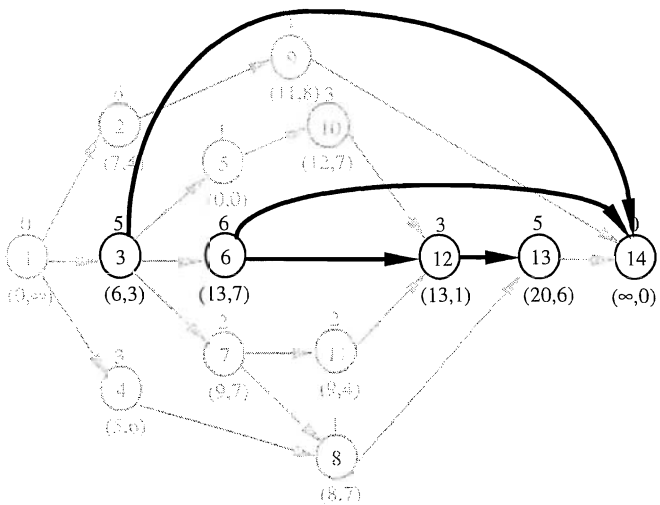

Figure 5. $DT = \{(3,14),(6,12),(12,13),(12,14)\}$

Although not the case in the example, it should be noted that a subtree of the due date tree which has an arc connected to the dummy start activity 1 can also be the subject of a recursive search. Although such subtrees have an arc connected to the dummy start activity which itself finishes at time zero, the recursive algorithm can indeed detect sets of activities *SA* which can be shifted further towards time zero.

Consider the AON project network given in Figure 6. There are 5 activities and two dummy activities, each with an activity duration denoted above the node and a due date and penalty cost denoted below the node.

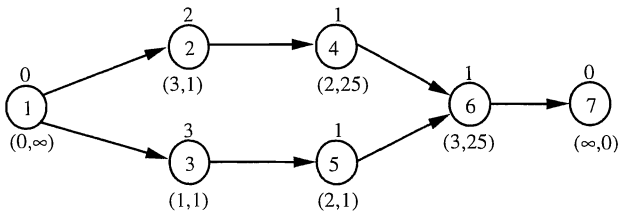

Figure 6. An example project network

Figure 7 displays the due date tree DT = {(1,3),(1,7),(2,4),(3,5),(4,6),(5,6)} after two shifts. The completion times of the activities are f_1=0, f_2=3, f_3=3, f_4=4, f_5=4, f_6=5 and f_7=5.

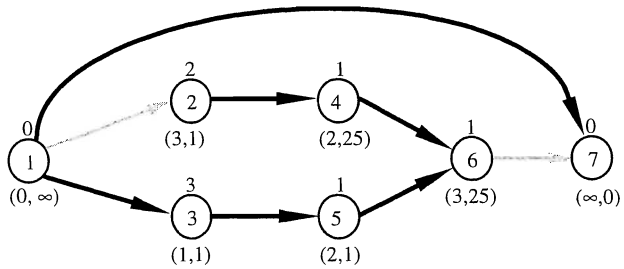

Figure 7. DT = {(1,3),(1,7),(2,4),(3,5),(4,6),(5,6)}

Activity 2 and activity 4 belong to a subtree which has an arc connected to the dummy start activity 1. However, the recursive search procedure is able to shift the set of activities SA = {2,4} towards time zero. The completion times of the activities are f_1=0, f_2=2, f_3=3, f_4=3, f_5=4, f_6=5 and f_7=5 and no further shift is possible. The weighted-earliness tardiness cost amounts to 80.

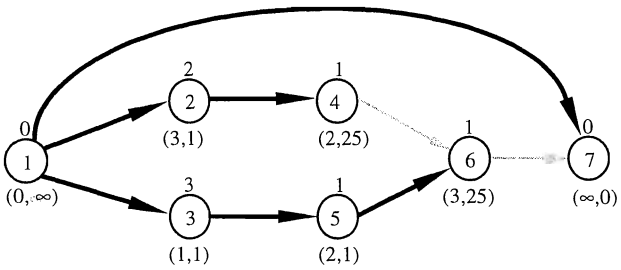

Figure 8. $DT = \{(1,3),(1,2),(1,7),(2,4),(3,5),(5,6)\}$

5. Computational experience

The recursive algorithm has been coded in Visual C++ Version 4.0 under Windows NT 4.0 on a Dell personal computer (Pentium 200 MHz processor). For the validation of the *WETPSP* we generated instances with *ProGen/Max* (Schwindt, 1995). These instances in activity-on-the-node format use four settings for the number of activities and three settings for the order strength *OS* as described in Table I. We then provided the problems with due dates and unit penalty costs. The due dates were generated as follows. First, we obtained a maximum due date of each project by multiplying the critical path length with a factor as given in Table I. We then randomly generated numbers between 1 and the maximum due date. Finally, we sorted these numbers and assigned them to the activities in increasing order, i.e. activity 1 is assigned the smallest due date, activity 2 the second smallest, etc.. Using seven settings for the due date generation and two settings for the unit penalty costs of the activities (both earliness and tardiness penalty cost), we obtained a dataset consisting of 1,680 instances.

Table I. Parameter settings used to generate the test instances

Number of activities	30, 60, 90 or 120
Order strength (*OS*) *(Mastor, 1970)*	0.25, 0.50 or 0.75
Due dates of the activities	randomly selected with factor 1.00, 1.25, 1.50, 1.75, 2.00, 2.25 or 2.50
Unit penalty cost	randomly selected from the interval [1,10] or [1,50]

Table II represents the average CPU-time and its standard deviation in milliseconds (actually, we have solved 1,000 replications for each problem and reported the time in seconds). Even instances with 120 activities can be solved within a very small amount of computation time. We should keep in mind that the unconstrained weighted earliness-tardiness project scheduling problem is probably not a goal by itself. Its solution may be used by a branch-and-bound procedure to compute bounds on the weighted earliness-tardiness cost of a resource-constrained weighted earliness-tardiness problem where the activities are also subject to renewable resource constraints (problem $m,1|cpm|early/tardy$). In that case the unconstrained problem should be solved efficiently in every (undominated) node of the branch-and-bound tree, which may run in the thousands (even millions). The reported CPU-times indicate that the recursive search procedure may well be used for that

end. Notice also the relatively small standard deviations, reflecting the rather robust behaviour of the procedure over the different problem instances.

Table II. Impact of the number of activities

# activities	# problems	Average CPU-time	Standard deviation
30	420	0.075	0.042
60	420	0.289	0.162
90	420	0.585	0.296
120	420	1.043	0.628

Table III shows a positive correlation between the *OS* of a project and the required CPU-time, i.e. the more dense the network, the more difficult the problem.

Table III. Impact of the order strength

OS	# problems	Average CPU-time	Standard deviation
0.25	560	0.403	0.378
0.50	560	0.478	0.440
0.75	560	0.613	0.647

Figure 9 illustrates the effect of the due date on the average required CPU-time. When the factor used for the due date generation is small, the problems contain many binding precedence relations and their solution will require an extensive search to shift many sets of activities *SA* to solve the problem. Problems with a large factor for the due date generation contain only few binding precedence relations in the due date tree. In that case, many activities will be scheduled on their due date and only a small number of shifts will be needed to solve the problem.

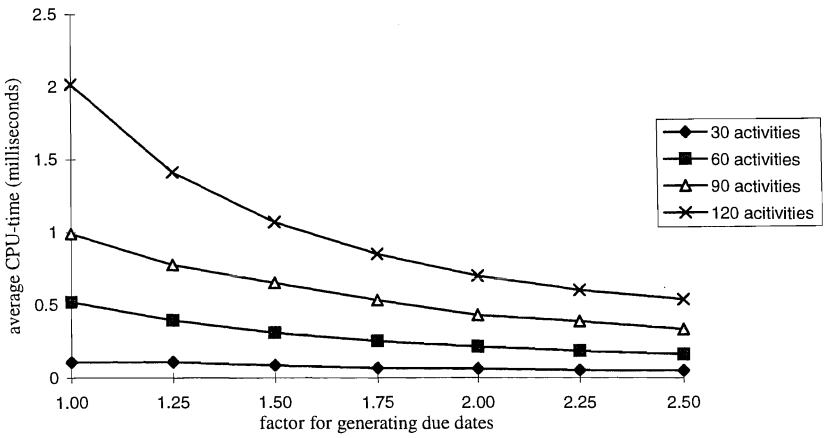

Figure 9. Effect of the due date

As expected, the earliness and tardiness penalty costs of the activities have no significant impact on the required CPU-time, as shown in Table IV.

Table IV. Impact of the unit penalty cost

Unit penalty cost	# problems	Average CPU-time	Standard deviation
[0,10]	840	0.495	0.508
[0,50]	840	0.501	0.510

6. Conclusions

In this paper an exact recursive search procedure was described for the unconstrained project scheduling problem with weighted earliness-tardiness penalty costs subject to zero-lag finish-start precedence constraints and in the absence of resource constraints (*cpm|early/tardy*). Each activity of this unconstrained project scheduling problem has a known deterministic due date, a unit earliness penalty cost as well as a unit tardiness penalty cost and the objective is to schedule the activities in order to minimize the weighted earliness-tardiness penalty cost of the project. With these features the problem becomes highly attractive in just-in-time environments. The exact procedure performs a forward pass calculation in order to schedule the activities such that they finish at their due date or later. Subsequently, a recursive search repetitively identifies those sets of activities for which a backward shift (towards time zero) decreases the weighted earliness-tardiness penalty cost of the project.

The procedure has been coded in Visual C++, version 4.0 under Windows NT 4.0 and has been tested on a randomly generated data set generated by *ProGen/Max* (Schwindt, 1995). The results of extensive computational tests obtained on a Dell personal computer (Pentium 200 MHz processor) reveal that the recursive algorithm is very efficient. This holds the promise that the procedure may be effectively used in branch-and-bound schemes for solving the *WETPSP* subject to resource constraints (*m*,1|*cpm|early/tardy*), which constitutes a promising area for future research.

References

De Reyck, B. and Herroelen, W., 1998, "An optimal procedure for the resource-constrained project scheduling problem with discounted cash flows and generalized precedence relations", *Computers and Operations Research*, 25, 1-17.

Herroelen, W., Van Dommelen, P. and Demeulemeester, E., 1997, "Project network models with discounted cash flows: A guided tour through recent developments", *European Journal of Operational Research*, 100, 97-121.

Herroelen, W., Demeulemeester, E. and De Reyck, B., 1998, "A classification scheme for project scheduling problems", in: Weglarz J. (Ed.), *Handbook on Recent advances in Project Scheduling*, Kluwer Academic Publishers, to appear.

Mastor, A.A., 1970, "An experimental and comparative evaluation of production line balancing techniques", *Management Science*, 16, 728-746.

Schwindt, C., 1995, "A new problem generator for different resource-constrained project scheduling problems with minimal and maximal time lags", WIOR-Report-449, Institut für Wirtschaftstheorie und Operations Research, University of Karlsruhe.