KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 0030

A VALIDATION OF PROCEDURES FOR MAXIMIZING THE NET PRESENT VALUE OF A PROJECT

by

M. VANHOUCKE

D/2000/2376/30

A VALIDATION OF PROCEDURES FOR MAXIMIZING THE NET PRESENT VALUE OF A PROJECT

Mario VANHOUCKE • Erik DEMEULEMEESTER • Willy HERROELEN

July 2000

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

A VALIDATION OF PROCEDURES FOR MAXIMIZING THE NET PRESENT VALUE OF A PROJECT

Mario Vanhoucke
mario.vanhoucke@econ.kuleuven.ac.be
Erik Demeulemeester
erik.demeulemeester@econ.kuleuven.ac.be
Willy Herroelen
willy.herroelen@econ.kuleuven.ac.be

Operations Management Group
Department of Applied Economics, Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven (Belgium)

ABSTRACT

The idea of maximizing the net present value of the cash flows of a project has gained increasing attention over the past decades. Several solution procedures have been presented in the literature to cope with these financial aspects of project management. In this paper we compare three solution procedures for the unconstrained project scheduling problem with discounted cash flows. Each activity of this unconstrained project scheduling problem has a known deterministic cash flow which can be negative, zero or positive. Progress payments and cash outflows occur at the completion of activities. The objective is to schedule the activities subject to the precedence constraints and a fixed deadline in order to maximize the net present value.

We compare two recursive search algorithms and a first-order steepest ascent approach for project scheduling problems where both minimal and maximal time-lags between the activities are considered. All procedures exploit the idea that positive cash flows should be scheduled as early as possible while negative cash flows should be scheduled as late as possible within the precedence constraints. The procedures have been coded in Visual C++, version 6.0 under Windows 2000 and have been validated on well-known problem sets.

Keywords: *Discounted cash flows; Recursive search; Steepest ascent approach.*

1 Introduction

In 1964, Battersby (1964) introduced the idea of maximizing the net present value (*npv*) of the cash flows of a project as a financially highly relevant criterion. Since then, a large amount of algorithms have been presented in the literature under different assumptions with respect to network representation (activity-on-the-node versus activity-on-the-arc) and cash flows patterns (positive and/or negative, event-oriented or activity-based and time-dependent vs. time-independent). In their paper Herroelen et al. (1997) examined the rationale behind this idea and gave an overview of the existing algorithms.

Recent research has focused on the case where activity cash flows are dependent on the completion times of the corresponding activities (e.g. the papers by Dayanand and Padman (1993a, 1993b, 1997), Etgar et al. (1996), Kazaz and Sepil (1996), Sepil and Ortaç (1997), Shtub and Etgar (1997), Etgar and Shtub (1999) and Vanhoucke et al. (1999b, 2000b)). In this paper, however, we restrict our attention to the maximization of the net present value of an unconstrained project in which the cash flows are independent of the completion times of the activities.

Depending on the type of precedence relations, we distinguish between procedures where only minimal time-lags between the activities are considered and problems with generalized precedence constraints, i.e. where both minimal and maximal time-lags are taken into account. Following the classification scheme of Herroelen et al. (1999), the first type of project scheduling problem can be categorized as $min, \delta_n, c_j \mid npv$ and is further denoted as the **max-*npv*** problem. Exact algorithms for the max-*npv* problem have been presented by Russell (1970), Grinold (1972), Elmaghraby and Herroelen (1990), Herroelen and Gallens (1993), Demeulemeester et al. (1996) and Vanhoucke et al. (1999a). The introduction of generalized precedence relations (*gpr*) transforms the problem into the **max-*npv-gpr*** problem (problem $gpr, \delta_n, c_j \mid npv$) and is the topic of research done by Kamburowski (1990), De Reyck and Herroelen (1996), Neumann and Zimmermann (1998) and Schwindt and Zimmermann (1998, 1999).

The organisation of the paper is as follows. In section 2 we describe the max-*npv* and max-*npv-gpr* problem. Section 3 reviews a number of algorithms from the literature. In section 4 we report the results of a computational experiment set up to validate the procedures described in this paper. We conclude in section 5 with some overall conclusions.

2 Description of the problem

The basic problem under study involves the scheduling of project activities in order to maximize the net present value (*npv*) of the project in the absence of resource constraints. Assume that 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 the precedence constraints. The activities are numbered from the dummy start activity 1 to the dummy end activity n. The duration of an activity is denoted by d_i ($1 \leq i \leq n$) and the performance of each activity involves a series of cash flow payments and receipts throughout the activity duration. Assume that cf_{it} ($1 < i < n$)

denotes the known deterministic cash flow of activity i in period t of its execution. A terminal value of each activity upon completion can be calculated by compounding the associated cash flow to the end of the activity as follows: $c_i = \sum_{t=1}^{d_i} cf_{it} e^{\alpha(d_i - t)}$, where α represents the discount rate and c_i the terminal value of cash flows of activity i at its completion. The nonnegative integer variables s_i and f_i ($1 \le i \le n$) denote the starting time and completion time, respectively, of activity i. The discounted value of activity i at the beginning of the project is $c_i e^{-\alpha(s_i + d_i)} = c_i e^{-\alpha f_i}$. A formulation of the problem can be given as follows:

$$\text{Maximize} \sum_{i=2}^{n-1} c_i e^{-\alpha(s_i + d_i)} \qquad [1]$$

Subject to

$$s_i + l_{ij} \le s_j \qquad \forall (i,j) \in A \qquad [2]$$

$$s_n \le \delta_n \qquad [3]$$

$$s_1 = 0 \qquad [4]$$

The objective in Eq. 1 maximizes the net present value of the project. The constraint set given in Eq. 2 maintains the precedence relations with time-lag l_{ij} among the activities. Eq. 3 limits the project duration to a negotiated project deadline δ_n and Eq. 4 forces the dummy start activity to start at time zero.

The time-lag l_{ij} and the different types of generalized precedence relations can be represented in a standardized form by reducing them to minimal start-start precedence relations as follows:

$$s_i + SS_{ij}^{\min} \le s_j \quad : \quad s_i + l_{ij} \le s_j \text{ with } l_{ij} = SS_{ij}^{\min}$$

$$s_i + SF_{ij}^{\min} \le f_j \quad : \quad s_i + l_{ij} \le s_j \text{ with } l_{ij} = SF_{ij}^{\min} - d_j$$

$$f_i + FS_{ij}^{\min} \le s_j \quad : \quad s_i + l_{ij} \le s_j \text{ with } l_{ij} = d_i + FS_{ij}^{\min}$$

$$f_i + FF_{ij}^{\min} \le f_j \quad : \quad s_i + l_{ij} \le s_j \text{ with } l_{ij} = d_i - d_j + FF_{ij}^{\min}$$

in the case of a minimal time-lag, and

$$s_i + SS_{ij}^{\max} \ge s_j \quad : \quad s_j + l_{ji} \le s_i \text{ with } l_{ji} = -SS_{ij}^{\max}$$

$$s_i + SF_{ij}^{\max} \ge f_j \quad : \quad s_j + l_{ji} \le s_i \text{ with } l_{ji} = d_j - SF_{ij}^{\max}$$

$$f_i + FS_{ij}^{\max} \ge s_j \quad : \quad s_j + l_{ji} \le s_i \text{ with } l_{ji} = -d_i - FS_{ij}^{\max}$$

$$f_i + FF_{ij}^{\max} \ge f_j \quad : \quad s_j + l_{ji} \le s_i \text{ with } l_{ji} = d_j - d_i - FF_{ij}^{\max}$$

in the case of a maximal time-lag (Bartusch et al., 1988).

In section 2.1 we give an overview of the literature for the max-*npv* problem. In section 2.2 we review the solution procedures for the max-*npv-gpr* problem.

2.1 The max-*npv* problem

Most of the exact solution procedures for solving the max-*npv* problem which have been offered in the literature take an event-oriented view with net cash flows associated with the events in AoA networks. Russell (1970) was the first to offer a nonlinear programming formulation and an exact solution procedure based on an iterative series of linear approximations and the solution of a transhipment problem over a network model. Grinold (1972) also takes an event-oriented view and shows that the problem can be transformed into an equivalent linear program. Elmaghraby and Herroelen (1990) have developed an exact solution procedure for AoA networks based on the intuitive argument that positive cash flows should be scheduled as early as possible and negative cash flows should be scheduled as late as possible within the precedence constraints. The algorithm operates by building tree structures in an iterative fashion and by determining proper displacement intervals for the trees. Herroelen and Gallens (1993) have streamlined the algorithm and report on favorable computational results on 250 randomly generated projects with a computer code written in the C language and running under the DOS operating system. Demeulemeester et al. (1996) propose an activity-oriented recursive search algorithm for the max-*npv* problem as described in Eqs. [1]-[4] with $l_{ij} = d_i$ for all $(i,j) \in A$. Vanhoucke et al. (1999a) have updated this recursive search algorithm and incorporated it in a branch-and-bound algorithm for the resource-constrained max-*npv* problem.

In section 3.1 we discuss the recursive search algorithm of Demeulemeester et al. (1996) and its adapted version of Vanhoucke et al. (1999a).

2.2 The max-*npv-gpr* problem

Kamburowski (1990) has presented an exact solution procedure for the max-*npv-gpr* problem based on the approach by Grinold (1972). The introduction of a new pivot rule extends this last procedure to generalized precedence relations. De Reyck and Herroelen (1996) have adapted the recursive search procedure of Demeulemeester et al. (1996) using the so-called distance matrix D in order to cope with generalized precedence relations. De Reyck and Herroelen (1998) have embedded this procedure into a branch-and-bound algorithm for the resource-constrained project scheduling problem with discounted cash flows and generalized precedence constraints. Neumann and Zimmermann (1998) have adapted the procedure by Grinold (1972) and have investigated different pivot rules. Finally, Schwindt and Zimmermann (1998, 1999) have presented a steepest ascent algorithm and compared different solution procedures on two randomly generated test sets.

In section 3.2 we describe the steepest ascent procedures proposed by Schwindt and Zimmermann (1998, 1999), while section 3.3 reviews the recursive search method by De Reyck and Herroelen (1996).

Table 1 gives an overview of the procedures which will be validated in this paper by means of a computational experiment.

Table 1. Overview of the procedures validated in this paper

The max-*npv* problem	The max-*npv-gpr* problem
1. Demeulemeester et al. (1996) and Vanhoucke et al. (1999a), further denoted as **the *recursive search method***	**1.** De Reyck and Herroelen (1996, 1998), further denoted as the ***generalized recursive search method***
2. Schwindt and Zimmermann (1998, 1999), further denoted as the ***steepest ascent approach***	**2.** An adapted version of Vanhoucke et al. (1999a) (i.e. the ***recursive search method***)
	3. Schwindt and Zimmermann (1998, 1999) (i.e. the ***steepest ascent approach***)

3 An overview of three procedures

In this section we discuss the recursive search method, the steepest ascent approach and the generalized recursive search method for maximizing the net present value of an unconstrained project under different types of precedence relations. We also propose some modifications of the recursive search method in order to improve its efficiency.

3.1 A recursive search procedure for the max-*npv* problem

Demeulemeester et al. (1996) have proposed a recursive search procedure for solving the max-*npv* problem with finish-start precedence relations with a time-lag of zero. The algorithm starts by calculating an *early tree* in which all activities are scheduled at their earliest completion times. In a second step, a *current tree CT* is build by delaying, in reverse order, all activities with a negative cash flow and no successor in the early tree as much as possible. In a third step the current tree is the subject of a *recursive search* (using the dummy start activity 1 as the search base) in order to identify sets of activities (*SA*) that might be shifted forward (away from time zero) to increase the net present value of the project. When a set of activities *SA* is found for which a forward shift leads to an increase of the *npv*, the algorithm computes an allowable displacement interval and updates the current tree *CT*. The completion times of the activities of *SA* are increased by the allowable displacement interval 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 *npv* are reported.

During the recursive search it is possible that the current tree disconnects into two parts. The first part is connected to the dummy start node and will be optimized by means of the recursive search (which uses the dummy start node as the search basis). This optimization stops when no set of activities can be shifted that increases the net present value. If this is the case, the algorithm optimizes the second part, which is connected to the deadline, by performing a similar recursion starting at the dummy end node until no set of activities can be found anymore. If this is the case, the algorithm stops and reports the finishing times.

Vanhoucke et al. (1999a) have refined the procedure by adding an extra arc from the dummy end node to the dummy start node. They can avoid the recursive step which

starts from the end node, combining an optimality guarantee with efficiency gains. The authors describe the recursive search algorithm by means of pseudocode and an illustrative example. The pseudocode of the third step, in which the recursion is repeated several times, can be written as follows (*CA* denotes the set of already considered activities, *CT* denotes the current tree, *DC* denotes the discounted cash flow and $v_{k^*l^*}$ the allowable displacement interval):

---

procedure Step 3 of the recursive search method;
CA = ∅;
Do *RECURSION*(1) → *SA'*, *DC'* (parameters returned by the recursive function);
Report the optimal completion times of the activities and the net present value *DC'*. STOP.

RECURSION(NEWNODE)
Initialize $SA = \{newnode\}$, $DC = DC_{newnode}$ and $CA = CA \cup \{newnode\}$;
Do $\forall i | i \notin CA$ and i succeeds *newnode* in the current tree *CT*:
RECURSION(*i*) → *SA'*, *DC'*
If $DC' \geq 0$ **then**
Set $SA = SA \cup SA'$ and $DC = DC + DC'$;
Else
CT=*CT*\(*newnode*,*i*);
Compute $v_{k^*l^*} = \min_{\substack{(k,l)\in A \\ k\in SA \\ l\notin SA}} \{s_l - d_k - s_k\}$ and set $CT=CT\cup(k^*,l^*)$;
Do $\forall j \in SA'$: set $s_j = s_j + v_{k^*l^*}$;
Go to STEP 3;
Do $\forall i | i \notin CA$ and i precedes *newnode* in the current tree *CT*:
RECURSION(*i*) → *SA'*, *DC'*;
Set $SA = SA \cup SA'$ and $DC = DC + DC'$;
Return;

The overall time complexity of the recursive search method still remains an open question. The recursion itself (which is a part of the recursive search method since the recursion can be re-executed several times) can be called upon itself at most n times (which corresponds to a search along the n-1 arcs of the current tree). At the end of the last recursion step, when a set of activities *SA* is found, an allowable displacement has to be calculated which requires $O(|A|)$ time. Consequently, the complexity of the recursion itself is $O(|A|)$ (or alternatively, $O(n^2)$ but the number of arcs $|A|$ is typically much smaller than n^2). The question how often the recursion is re-executed, however, remains open.

In section 3.4 we show how a simple modification can be made in order to solve the max-*npv*-*gpr* problem by means of the recursive search method as described in this section.

3.2 A steepest ascent procedure for the max-*npv-gpr* problem

Schwindt and Zimmermann (1998, 1999) propose two steepest ascent procedures for solving the project scheduling problem subject to temporal constraints in order to maximize the net present value. In this section we describe the steepest ascent procedure of Schwindt and Zimmermann (1999), clearly the best performing algorithm.

The algorithm starts with building a spanning tree in which all activities are scheduled at their earliest start schedule. In a second step, a steepest ascent direction is determined in an O(n) time implementation by finding sets of activities which can be shifted forward in time in order to increase the net present value. The actual shift occurs in a third step (with time complexity O($|A|$ *log* $|A|$)) which searches for a destination schedule S' such that the net present value of the new schedule S' is greater than or equal to the old schedule S. The algorithm repeats step 2 and continues until no steepest ascent direction can be found. As is the case with the recursive search method, the overall time complexity of the algorithm is still unknown.

In their description of the algorithm, the authors use ϕ_i to denote the partial derivative of the discounted value of c_i with respect to the finishing time f_i, i.e. $\frac{d(c_i e^{-\alpha(s_i+d_i)})}{d(s_i+d_i)} = \frac{d(c_i e^{-\alpha f_i})}{df_i} = -\alpha c_i e^{-\alpha f_i} = -\alpha c_i e^{-\alpha(s_i+d_i)}$. This implies that an activity for which $\phi_i>0$ has a negative cash flow and can possibly be delayed.

In order to find a steepest ascent direction (step 2), the authors search for activities to be merged as follows: an activity i with at most one successor j in the spanning tree can be merged to an aggregate activity with partial derivative $\phi_i + \phi_j$. Moreover, an activity j with exactly one predecessor i in the spanning tree can be merged in the same way. If, however, this activity j has a partial derivative $\phi_j > 0$ then it is added to the set of possibly delayed nodes Z (instead of being merged). The algorithm performs these steps until all nodes (except node 1) have been deleted from the spanning tree. The pseudocode of the steepest ascent procedure is written below. The node set $C(j)$ contains node j and all other nodes merged with j.

STEEPEST ASCENT DIRECTION

Initialize $Z = \emptyset$ and $V = N$;

$\forall i \in N$ set $C(i) = \{i\}$ and $\phi_i = -\alpha c_i e^{-\alpha(s_i+d_i)}$;

While $V \neq \{1\}$

If V includes a node $i \neq 1$ with $P_i \cap V = \emptyset$ possessing at most one successor $j \in V$ **then**

$V = V\backslash\{i\}$, $\phi_j = \phi_j + \phi_i$ and $C(j) = C(j) \cup C(i)$;

Else

Determine node $j \in V$, $j \neq 1$ with $S_j \cap V = \emptyset$ possessing exactly one predecessor $i \in V$;

$V = V\backslash\{j\}$;

If $\phi_j > 0$ **then** $Z = Z \cup C(j)$;

Else $\phi_i = \phi_i + \phi_j$ and $C(i) = C(i) \cup C(j)$;

Return;

Whenever the algorithm has determined a steepest ascent direction with its corresponding set of possibly delayed activities Z, the algorithm searches for a new destination schedule which improves the net present value (step 3). Therefore, the algorithms calculates a displacement interval, shifts the activities of Z and updates the spanning tree ST and the node set Z. This algorithm is repeated until all nodes are removed from Z. The pseudocode of this step is given below. Remark that the node set $C(j)$ of the weak component of the spanning tree ST with $k \in C(j)$ simply refers to the set of nodes which has to be delayed. This refers to the maximal set of nodes $C(j)$ to which node k belongs, i.e. the node set $C(j) | k \in C(j)$ for which $\neg \exists l \in N | C(j) \subset C(l)$.

VERTEX ASCENT (FINDING A NEW DESTINATION SCHEDULE)

Do $\forall (i,j) \in ST | j \notin C(i)$ and $i \notin C(j) : ST = ST \backslash (i,j)$;

While $Z \neq \varnothing$

Compute $v_{k^*l^*} = \min_{\substack{(k,l) \in A \\ k \in Z \\ l \notin Z}} \{ s_l - s_k - l_{kl} \}$;

Determine the node set $C(j)$ of the *weak component* of ST with $k \in C(j)$;

Do $\forall i \in C(j) : s_i = s_i + v_{k^*l^*}$;

$Z = Z \backslash C(j)$;

$ST = ST \cup (k,l)$;

Return;

The approach is quite analogous to the recursive search method as described in section 3.1. The spanning tree corresponds to the early tree in the recursive search method approach. In the recursive search method, both step 2 (finding a steepest ascent direction) and step 3 (finding a new destination schedule) are performed in one recursive step. However, the recursion searches for a steepest ascent direction along the arcs in the current tree, which is far more efficient than the iterative steepest ascent procedure presented in this section. On the other hand, the last mentioned procedure identifies the activities to be delayed in one step and delays them one by one in a second step. This is clearly more efficient than the approach of the recursive search method in which the recursive search is repeated each time a set of activities is found and delayed. This observation was an incentive to combine these two approaches by slightly adapting the recursive search procedure. This is the subject of section 3.4.

3.3 A generalized recursive search method for the max-*npv-gpr* problem using a distance matrix

De Reyck and Herroelen (1996, 1998) have adapted the recursive search algorithm of Demeulemeester et al. (1996) to cope with generalized precedence relations. They calculate a distance matrix D using the Floyd-Warshall algorithm (Lawler, 1976) which serves as a basis for the current tree in the recursive search. As is the case for the recursive search method for the max-*npv* problem, the procedure starts by calculating an early tree which spans all activities at their earliest start time. The current tree calculation - which possibly has to be repeated several times - delays, in reverse order, the activities with a negative cash flow and no successor in the early

tree. Finally, a recursion step delays sets of activities SA in order to increase the net present value of the project. To that purpose, an allowable displacement interval $v_{k^*l^*}$ has to be calculated with $v_{k^*l^*} = \min_{\substack{(k,l)\in A \\ k\in SA \\ l\notin SA}} \{d_{1,l} - d_{1,k} - d_{k,l}\}$, with $d_{i,j}$ the element of row i and column j of the distance matrix D. Notice that this allowable displacement interval searches for a new arc based on the distance matrix D instead of the original network.

During the calculation of the current tree it is possible that the delay of an activity i allows for an additional delay of another activity j ($j>i$) which was already considered in the current tree calculation. Therefore, when an activity has been delayed in the current tree, this step has to be repeated. In the next example, we will show that this can possibly lead to nodes which are no longer connected with the dummy start node (and consequently, which can miss the optimum since these nodes are no longer the subject of the recursive search).

The example network of Fig. 1 contains 4 non-dummy activities and minimal and maximal time-lags between the activities. Again, the activity duration is displayed above the node while the cash flow is given below the node. The discount rate is 0.016 and the project deadline δ_n is 10. The numbers associated with the arcs denote the generalized precedence relations. A minimal time-lag is denoted by an arc (i,j) with a positive number while a maximal time-lag is represented by an arc (j,i) with a negative number (e.g. arc (4,3)). For the sake of simplicity, all arcs are of the start-start (SS) type.

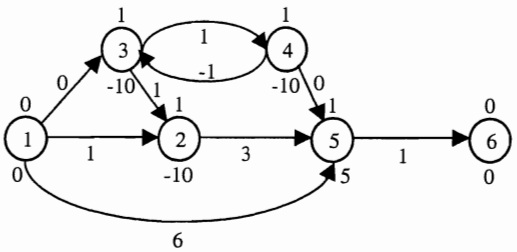

Fig. 1. An example network for the max-*npv-gpr* problem

The early tree, which spans all activities at its earliest start schedule $s_1 = 0$, $s_2 = 1$, $s_3 = 0$, $s_4 = 1$, $s_5 = 6$ and $s_6 = 7$, and its corresponding distance matrix D is given in Fig. 2.

$$D = \begin{pmatrix} 0 & 1 & 0 & 1 & 6 & 7 \\ -\infty & 0 & -\infty & -\infty & 3 & 4 \\ -\infty & 1 & 0 & 1 & 4 & 5 \\ -\infty & 0 & -1 & 0 & 3 & 4 \\ -\infty & -\infty & -\infty & -\infty & 0 & 1 \\ -\infty & -\infty & -\infty & -\infty & -\infty & 0 \end{pmatrix}$$

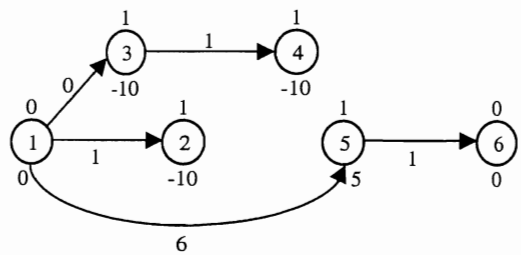

Fig. 2. The early tree of the example network for the max-*npv-gpr* problem

During the current tree calculation, all activities with negative cash flows and no successor in the early tree, are shifted towards the deadline. First of all, activity 4 is delayed by removing arc (3,4) in the current tree and adding arc (4,2) (i.e. $d_{1,2} - d_{1,4} - d_{4,2} = 1 - 1 - 0 = 0$ is minimal). Then, the algorithm removes arc (1,3) and adds arc (3,2) in the current tree. Last, activity 2 is delayed by removing the two previously added arcs, (3,2) and (4,2), and adding arc (2,5) in the current tree. Activities 3 and 4 are now no longer connected with the dummy start node. The current tree is now repeated for the second time, in which activity 4 is now delayed by adding arc (4,3) in the current tree. The current tree calculation stops and the algorithm finds the current tree as displayed in Fig. 3. Since activities 3 and 4 are now no longer connected with the dummy start node, the generalized recursive search will fail to find the exact solution.

$$D = \begin{pmatrix} 0 & \mathbf{3} & 0 & 1 & 6 & 7 \\ -\infty & 0 & -\infty & -\infty & 3 & 4 \\ -\infty & 1 & 0 & 1 & 4 & 5 \\ -\infty & 0 & -1 & 0 & 3 & 4 \\ -\infty & -\infty & -\infty & -\infty & 0 & 1 \\ -\infty & -\infty & -\infty & -\infty & -\infty & 0 \end{pmatrix}$$

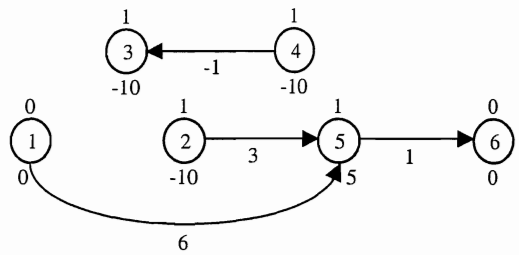

Fig. 3. The erroneous current tree of the example network for the max-*npv-gpr* problem

In order to test the efficiency of this procedure (see section 4), we have coded it in Visual C++ without the erroneous calculation of the current tree.

3.4 Adaptations to the recursive search method

In order to gain efficiency we have implemented three modifications in the recursive search method. The first modification combines a forward and backward procedure in order to reduce the number of displacements. Secondly, we alter the displacement conditions to cope with generalized precedence relations. A third modification combines the idea of the recursive search with the displacement approach of the steepest ascent procedure.

3.4.1 Forward and backward

In their paper, Vanhoucke et al. (1999a) have tested the impact of the percentage of negative cash flows in the project on the required CPU-time and have revealed that the higher the percentage (except for 100% negative cash flows) of negative cash flows, the more difficult the problem. This is quite logic since a higher number of negative cash flows implies a larger number of shifts and consequently a more extensive search in the current tree. We therefore use an alternative approach for the original recursive search method when more than 50% of the activities have a negative cash flow. In this case, we schedule the activities as late as possible (within the project deadline) and try to find sets of activities to shift backward (toward time

zero) in order to increase the net present value. We simply use the dummy end node as a basis of our recursive search instead of the dummy start node. This will lead to a considerable reduction of the number of shifts in our search.

3.4.2 Adapting the recursive search procedure for the max-*npv-gpr* problem

The recursive search method can be easily modified in order to cope with generalized precedence relations. To that purpose, we have to adapt the calculation of the early tree and the allowable displacement interval.

The early tree *ET* simply schedules all the activities as soon as possible within the precedence constraints (comparable with the original recursive method). Using *bool* as a boolean variable used in the while test, the pseudocode to build the early tree *ET* is as follows:

BUILD EARLY TREE
Initialize $CA = \{1\}$, $ET = \varnothing$ and *bool* = true;
Set $s_1 = 0$ and $s_i = -\infty | i \in N \backslash \{1\}$;
While *bool* = true
bool = false;
Do $\forall (i,j) \in A$
If $s_i + l_{ij} > s_j$ **then** *bool* = true and $s_j = s_j + l_{ij}$;
While $CA \neq N$
Do $\forall (i,j) \in A$
If $i \in CA$ and $j \notin CA$ and $s_i + l_{ij} = s_j$ **then**
$CA = CA \cup \{j\}$;
$ET = ET \cup (i,j)$;
Return;

The allowable displacement interval is $\min_{\substack{(k,l)\in A \\ k\in SA \\ l\notin SA}} \{s_l - s_k - l_{kl}\}$ which simply calculates the minimal distance over which an activity $k \in SA$ can be shifted until it connects with an activity $l \notin SA$. Remark that De Reyck and Herroelen (1996) uses the distance matrix *D* as a basis for the tree calculation while our approach uses the original precedence relations as a basis for the early tree. In doing so, we avoid the time-consuming computation of this distance matrix.

3.4.3 Combining the recursive search method with the steepest ascent approach

In section 3.2 we claimed that the steepest ascent approach is almost identical to the recursive search method. In the recursive search method, each time a set of activities *SA* has been found which can be shifted forward, a number of steps are successively performed. First, the allowable displacement is calculated and the activities are shifted. Second, the tree is updated and finally, the recursion step is repeated. The steepest ascent approach calculates all sets of activities in one step and shifts them one by one in a second step.

The combined approach which is the subject of this section simply divides the logic of the recursive search method into two parts. The first part (still denoted as the *RECURSION* step) only searches for sets of activities *SA* with a negative *npv* and which are, consequently, candidates to be shifted. We therefore use *SS* to denote the saved sets *SA* found in the recursion step. The second step (referred to as the *Shift_activities* step) calculates an allowable displacement interval for the sets $SA \in SS$ and is almost identical to the vertex ascent procedure of section 3.2. Notice that the calculation of this displacement interval was originally included in the recursion step of Vanhoucke et al. (1999a) (see section 3.1).

The pseudocode of a combined approach is described hereafter. We use *SA* to denote a set of activities, *SS* to denote the saved sets *SA* found in the recursion step, *CA* to denote the set of already considered activities and *DC* to denote the discounted cash flows.

---

procedure Rec_New;
  $CA = SS = \emptyset$;
  Do *RECURSION*(1)
  If $SS \neq \emptyset$ **then** *Shift_activities()* and **repeat** *RECURSION*(1)
  Else Report the optimal completion times of the activities and the *npv DC'*. STOP.

---

with the pseudocode of the recursion step as follows:

RECURSION(NEWNODE)
  Initialize $SA = \{newnode\}$, $DC = DC_{newnode}$ and $CA = CA \cup \{newnode\}$;
  Do $\forall i | i \notin CA$ and i succeeds *newnode* in the early tree *ET*:
    RECURSION$(i) \rightarrow SA', DC'$
    If $DC' \geq 0$ **then** Set $SA = SA \cup SA'$ and $DC = DC + DC'$;
    Else $ET = ET \backslash (newnode, i)$ and $SS = SS \cup SA'$;
  Do $\forall i | i \notin CA$ and i precedes *newnode* in the early tree *ET*:
    RECURSION$(i) \rightarrow SA', DC'$;
    Set $SA = SA \cup SA'$ and $DC = DC + DC'$;
Return;

---

and the pseudocode of the delaying step as follows:

SHIFT_ACTIVITIES(*SS*)
  Let $Z = \{i \in SA | SA \in SS\}$ be the set of activities which can possibly be delayed;
  While $Z \neq \emptyset$ **do**
    Compute $v_{k^*l^*} = \min_{\substack{(k,l) \in A \\ k \in Z \\ l \notin Z}} \{s_l - s_k - l_{kl}\}$;
    Do $\forall i \in SA | k^* \in SA$: Set $s_i = s_i + v_{k^*l^*}$ and $Z = Z \backslash \{i\}$;
    Set $ET = ET \cup (k^*, l^*)$;
Return;

In the next section we compare the different solution procedures by means of a computational experiment on two well-known datasets taken from the literature.

4 Computational experience

In order to test the efficiency of all procedures, we have coded them in Visual C++ version 6.0 under Windows 2000 on a Dell personal computer, Pentium III, 800 MHz processor. In order to validate the procedures for the max-*npv* procedure, we have downloaded the 600 well-known problem instances with 120 activities from the *PSPLIB* developed by Kolisch and Sprecher (1996) at *http://www.bwl.uni-kiel.de/Prod/psplib/index.html*. Moreover, we have used the 1,440 instances from De Reyck (1998) to validate the procedures for the max-*npv-gpr* problem. We have extended both datasets with cash flows generated from the interval [-500;500], a discount rate of 1.6% and a project deadline which equals the critical path length increased by 100.

In the sequel of this section we use "% neg" to denote the percentage of negative cash flows in the problem instances and "*Rec* 1" to refer to the original recursive search method as proposed by Demeulemeester et al. (1996) and modified by Vanhoucke et al. (1999a). "*Rec* 2" and "*Rec* 3" refer to the adaptations as proposed in section 3.4.1 and section 3.4.3 respectively. "*SA*" refers to the procedure of Schwindt and Zimmermann (1999). We make use of the extension "*gpr*" in order to refer to the same procedures for the max-*npv-gpr* problem. The abbreviation "*DRH*" is used to refer to the procedure of De Reyck and Herroelen (1996) (without the calculation of the current tree).

4.1 Computational experience for the max-*npv* problem

Table 2 reports the average CPU-time in milliseconds for the 6,600 instances (600 instances and 11 settings for the percentage of negative cash flows) for the max-*npv* problem. This table reveals that the recursive procedure (*Rec* 1) outperforms the procedure by Schwindt and Zimmermann (1999). Since the procedure *Rec* 1 is positively correlated with the percentage of negative cash flows in the problem instances, we prefer to use the approach of *Rec* 2. In this approach, problem instances can be resolved by using a backward recursive search procedure whenever the percentage of negative cash flows exceeds 50%. The combined approach (*Rec* 3 is a combination of *Rec* 1 and *SA*) performs best, as shown in Table 2. This is due to the two aforementioned reasons: the recursion *Rec* 1 searches for sets of activities along the arcs in the current tree while the steepest ascent approach *SA* delays the set of activities one by one in the same step.

Table 2. Average CPU-time in milliseconds for the max-*npv* problem

% neg	*Rec* 1	*Rec* 2	*Rec* 3	*SA*
0%	0,059	0,059	0,059	0,126
10%	0,136	0,136	0,139	0,375
20%	0,188	0,188	0,164	0,438
30%	0,358	0,359	0,231	0,612
40%	0,554	0,554	0,314	0,845
50%	1,062	1,064	0,483	1,354
60%	1,339	0,568	0,456	1,340
70%	1,691	0,317	0,491	1,428
80%	1,651	0,202	0,517	1,250
90%	1,733	0,128	0,579	1,305
100%	0,067	0,059	0,068	0,840
	0,804	**0,330**	**0,318**	**0,901**

In Table 3 we display the average and maximum number of iterations for the max-*npv* problem. For the recursion procedures, the number of iterations is defined as the number of times the recursion is called. For the steepest ascent approach, the number of iterations equals the number of times a steepest ascent direction has to be looked for. Since in this approach (and in the adapted recursion *Rec* 3) the algorithm searches for all steepest ascent directions at one time before any shift is made, the number of iterations is dramatically lower than in the original recursion procedure.

Table 3. Average and maximum number of iterations for the max-*npv* problem

% neg	*Rec* 1	*Rec* 2	*Rec* 3	*SA*
0%	1,00	1,00	1,00	1,00
10%	3,97	3,97	2,51	3,12
20%	6,12	6,12	2,91	3,53
30%	13,64	13,64	3,86	4,47
40%	22,66	22,66	4,91	5,54
50%	47,22	47,22	6,77	7,42
60%	63,80	23,76	5,89	6,85
70%	88,60	11,93	5,85	6,71
80%	90,74	6,71	6,31	5,33
90%	99,71	3,66	6,98	4,95
100%	1,00	1,00	1,00	2,00
Avg	39,86	12,88	4,36	4,63
Max	278	126	19	19

4.2 Computational experience for the max-*npv-gpr* problem

Table 4 reports the average CPU-time in milliseconds for the 15,840 (1,440 instances and 11 settings for the percentage of negative cash flows) instances for the max-*npv-gpr* problem. The results are quite similar to the results in section 4.1. The recursive search method, on the one hand, outperforms both the steepest ascent procedure and the generalized recursive search method (*DRH*). The steepest ascent procedure, on the other hand, of Schwindt and Zimmermann (1999) outperform the *DRH* procedure. This is due to the fact that the latter needs the very time-consuming calculation of the

distance matrix D in order to solve the problem. As is the case for the max-*npv* problem, the combined approach (*Rec* 3) performs best for solving the max-*npv-gpr* problem to optimality.

Table 4. Average CPU-time in milliseconds for the max-*npv-gpr* problem

% neg	*Rec_gpr* 1	*Rec_gpr* 2	*Rec_gpr* 3	*SA_gpr*	*DRH*
0%	1,109	1,110	1,107	1,372	29,735
10%	1,164	1,163	1,160	1,720	34,493
20%	1,205	1,205	1,182	1,853	37,350
30%	1,313	1,315	1,212	1,959	43,561
40%	1,492	1,494	1,248	2,000	49,076
50%	1,735	1,739	1,294	2,038	52,181
60%	1,941	1,552	1,334	2,078	53,024
70%	2,159	1,435	1,385	2,120	53,612
80%	2,356	1,369	1,447	2,138	51,848
90%	2,492	1,295	1,493	2,107	50,008
100%	2,643	1,225	1,532	1,982	47,201
	1,783	**1,355**	**1,309**	**1,943**	**45,644**

A same conclusion can be drawn for the number of iterations for the max-*npv-gpr* problem which is displayed in Table 5.

Table 5. Average and maximum number of iterations for the max-*npv-gpr* problem

% neg	*Rec_gpr* 1	*Rec_gpr* 2	*Rec_gpr* 3	*SA_gpr*	*DRH*
0%	1,00	1,00	1,00	1,00	1,00
10%	3,14	3,14	2,22	2,22	34,50
20%	4,93	4,93	2,67	2,67	53,12
30%	9,62	9,62	3,14	3,14	93,01
40%	19,07	19,07	3,45	3,45	139,03
50%	36,98	36,98	3,54	3,54	185,22
60%	55,48	15,59	3,46	3,46	212,54
70%	79,44	10,09	3,29	3,29	234,11
80%	102,51	7,14	3,05	3,05	238,58
90%	121,55	3,87	2,68	2,68	233,63
100%	141,48	1,00	2,00	2,00	224,81
Avg	**52,29**	**10,22**	**2,77**	**2,77**	**149,96**
Max	**362**	**117**	**18**	**18**	**942**

5 Conclusions

In this paper we have compared three solution procedures for the unconstrained project scheduling problems with discounted cash flows with minimal (max-*npv* problem) and generalized precedence constraints (max-*npv-gpr* problem).

For the max-*npv* problem we compared the recursive search procedure (Demeulemeester et al., 1996 and Vanhoucke et al., 1999a) with a steepest ascent procedure (Schwindt and Zimmermann, 1999). Computational experience has revealed that the recursive search method outperforms the steepest ascent procedure. Moreover, a combination of both approaches results in the lowest CPU times.

Similar results have been found for the max-*npv-gpr* problem. Additional tests have shown that both the recursive search procedure and the steepest ascent procedure outperform the procedure by De Reyck and Herroelen (1996). The main reason is that the distance matrix is a very time-consuming activity in the last mentioned procedure.

A similar comparison can be made for the recursive search method for the weighted earliness-tardiness project scheduling problem of Vanhoucke et al. (2000a) and the steepest descent procedure of Schwindt (1999).

Acknowledgements

We would like to thank Dr. Christoph Schwindt for providing us useful information about the steepest ascent procedure. We are also grateful to Dr. Bert De Reyck for his helpful comments with respect to the procedure for the max-*npv-gpr* problem.

References

Battersby, A., 1964, "Network analysis for planning and scheduling", MacMillan.

Bartusch, M, Möhring, R.H. and Radermacher, F.J., 1988, "Scheduling project networks with recource constraints and time windows", *Annals of Operations Research*, 16, 201-240.

Dayanand, N. and Padman, R., 1993a, "The payment scheduling problem in project networks", Working Paper 9331, The Heinz School, CMU, Pittsburgh, PA 15213, U.S.A.

Dayanand, N. and Padman, R., 1993b, "Payments in projects: a constractor's model", Working Paper 9371, The Heinz School, CMU, Pittsburgh, PA 15213, U.S.A.

Dayanand, N. and Padman, R., 1997, "On modeling payments in project networks", *Journal of the Operational Research Society*, 48, 906-918.

Demeulemeester, E., Herroelen, W. and Van Dommelen, P., 1996, "An optimal recursive search procedure for the deterministic unconstrained max-*npv* project scheduling problem", Research Report 9603, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium.

De Reyck, B., 1998, "Scheduling Projects with Generalized Precedence Relations - Exact and Heuristic Procedures", Ph.D. Dissertation, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium.

De Reyck, B. and Herroelen, W., 1996, "An optimal procedure for the unconstrained max-*npv* project scheduling problem with generalized precedence relations", Research Report 9642, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium.

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.

Elmaghraby, S.E. and Herroelen, W., 1990, "The scheduling of activities to maximize the net present value of projects", *European Journal of Operational Research*, 49, 35-49.

Etgar, R., Shtub, A. and LeBlanc, L.J., 1996, "Scheduling projects to maximize net present value - The case of time-dependent, contingent cash flows", *European Journal of Operational Research*, 96, 90-96.

Etgar, R. and Shtub, A., 1999, "Scheduling project activities to maximize the net present value - The case of linear time dependent, contingent cash flows", *International Journal of Production Research*, 37, 329-339.

Grinold, R.C., 1972, "The payment scheduling problem", *Naval Research Logistics Quarterly*, 19, 123-136.

Herroelen, W. and Gallens, E., 1993, "Computational experience with an optimal procedure for the scheduling of activities to maximize the net present value of projects", *European Journal of Operational Research*, 65, 274-277.

Herroelen, W., Demeulemeester, E. and Van Dommelen, P., 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., 1999, "A classification scheme for project scheduling problems", in: Weglarz J. (Ed.), *Handbook on Recent Advances in Project Scheduling*, Kluwer Academic Publishers, Chapter 1, 1-26.

Kamburowski, J., 1990, "Maximizing the project net present value in activity networks under generalized precedence relations", Proceeding of 21st DSI Annual meeting, San Diego, 748-750.

Kazaz, B. and Sepil, C., B., 1996, "Project scheduling with discounted cash flows and progress payments", *Journal of the Operational Research Society*, 47, 1262-1272.

Kolisch, R. and Sprecher, A., 1996, "PSPLIB - A project scheduling library", *European Journal of Operational Research*, 96, 205-216.

Lawler, E.L., 1976, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York.

Neumann, K. and Zimmermann, J., 1998, "Exact and heuristic procedures for net present value and resource levelling problems in project scheduling", WIOR-Report-538, Institut für Wirtschaftstheorie und Operations Research, University of Karlsruhe, Germany.

Russell, A.H., 1970, "Cash flows in networks", *Management Science*, 16, 357-373.

Schwindt, C., 1999, "Minimizing earliness-tardiness costs of resource-constrained projects", Submitted to: Operations Research Proceedings 1999, Symposium on Operations Research 99, Magdeburg, Germany.

Schwindt, C. and Zimmermann, J., 1998, "Maximizing the net present value of projects subject to temporal constraints", WIOR-Report-536, Institut für Wirtschaftstheorie und Operations Research, University of Karlsruhe, Germany.

Schwindt, C. and Zimmermann, J., 1999, "A steepest ascent approach to maximizing the net present value of projects", WIOR-Report-576, Institut für Wirtschaftstheorie und Operations Research, University of Karlsruhe, Germany.

Sepil, C. and Ortaç, N., 1997, "Performance of the heuristic procedures for constrained projects with progress payments", *Journal of the Operational Research Society*, 48, 1123-1130.

Shtub, A. and Etgar, R., 1997, "A branch-and-bound algorithm for scheduling projects to maximize net present value: the case of time dependent, contingent cash flows", *International Journal of Production Research*, 35, 3367-3378.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W., 1999a, "On maximizing the net present value of a project under resource constraints", Research Report 9915, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W., 1999b, "Scheduling projects with linear time-dependent cash flows to maximize the net present value", Research Report 9949, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W., 2000a, "An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem", to appear in *Annals of Operations Research*.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W., 2000b, "Maximizing the net present value of a project with progress payments", Research Report 0028, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium.