DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9603

An Optimal Recursive Search Procedure for the Deterministic Unconstrained Max-npv Project Scheduling Problem

by

Erik DEMEULEMEESTER

Willy HERROELEN

Patrick VAN DOMMELEN

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

AN OPTIMAL RECURSIVE SEARCH PROCEDURE FOR THE DETERMINISTIC UNCONSTRAINED MAX-NPV PROJECT SCHEDULING PROBLEM

Erik L. DEMEULEMEESTER • Willy S. HERROELEN • Patrick VAN DOMMELEN

Katholieke Universiteit Leuven
Department of Applied Economics
Naamsestraat 69
B-3000 Leuven (Belgium)
Tel 32-16-32 69 70 or 32-16-32 69 72
Fax 32-16-32 67 32
e-mail: erik.demeulemeester@econ.kuleuven.ac.be; willy.herroelen@econ.kuleuven.ac.be

December 1995

This research was sponsored by the National Science Foundation under Research Contract FKFO N° 2.0051.94

An Optimal Recursive Search Procedure for the Deterministic Unconstrained Max-npv Project Scheduling Problem

Erik Demeulemeester • Willy Herroelen • Patrick Van Dommelen
Department of Applied Economics, Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven (Belgium)

ABSTRACT

The deterministic unconstrained max-npv problem involves the scheduling of the activities of a project in order to maximize its net present value. Projects are assumed to be represented by an activity network in activity-on-the-node format. Activities have a known deterministic duration and are subject to zero-lag finish-start precedence constraints. Cash flows are assumed to be known in both their amount and timing. Progress payments and cash outflows are made upon the completion of certain activities. The project is to be scheduled subject to a fixed deadline in the absence of resource constraints. We describe a new optimal procedure which performs a recursive search on partial tree structures based on the fundamental logic 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 procedure has been coded in Visual C++, version 2.1 for use on a personal computer. Encouraging computational results obtained on two data sets confirm that the search procedure outperforms an adapted version of the procedure which was previously developed by Grinold for projects in activity-on-the-arc format.

(Project Scheduling - Discounted cash flows; Net Present Value; Optimal search)

1. Introduction

In recent years, a number of publications have dealt with the project scheduling problem under the objective of maximizing the net present value *(npv)* of the project. The majority of the contributions assume a completely *deterministic* project setting, in which all relevant problem data, including the various cash flows, are assumed known from the outset. Research efforts have led to optimal procedures for the *unconstrained* project scheduling problem, where activities are only subject to precedence constraints. In addition, numerous efforts aim at providing optimal or suboptimal solutions to the project scheduling problem under various types of resource *constraints*, using a rich variety of often confusing assumptions with respect to network representation (activity-on-the-node versus activity-on-the-arc), cash flow patterns (positive and/or negative, event-oriented or activity-based), and resource constraints (capital constrained, different resource types, materials considerations, time/cost trade-offs). A number of efforts focus on the simultaneous determination of both the amount and timing of payments. Last, a modest start has been taken in tackling the *stochastic* aspects of the scheduling problem involved. For a recent review of the vast literature and a categorization of the solution procedures, we refer the reader to Herroelen et al. (1995).

In this paper we focus on the *deterministic unconstrained max-npv problem.* All optimal procedures for solving this problem (Russell 1970, Grinold 1972, Elmaghraby and Herroelen 1990 and Herroelen and Gallens 1993) have originally been developed for activity-on-the-arc (AoA) networks and take an event-oriented view in which both positive and negative cash flows occur as events (nodes) in the project are realized. In this paper we take an *activity-oriented* view in which positive and/or negative cash flows are associated with the completion of activities (nodes) in an *activity-on-the-node (AoN)* network. Activities have a known deterministic duration and are subject to zero-lag finish-start precedence constraints (the procedure can easily be extended to include precedence diagramming type of precedence constraints). Cash flows are assumed to be known in both their amount and timing. Progress payments and cash outflows are made upon the completion of certain activities. The project is to be scheduled subject to a fixed deadline in the absence of resource constraints.

The organization of the paper is as follows. In §2 we offer a formulation of the unconstrained deterministic max-npv problem. In §3 we describe a new optimal procedure and illustrate it on a problem example. In §4 we present detailed computational results on two problem sets. §5 is then reserved for our overall conclusions.

2. The deterministic unconstrained max-npv problem

The deterministic unconstrained max-npv problem involves the scheduling of project activities in order to maximize the net present value *(npv)* of the project. The project is represented by an 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. We assume, without loss of generality, that there is a single dummy start node *1* and a single dummy end node $n = |N|$. The problem is unconstrained in that no constraints are imposed on the use of resources. The activities have a fixed duration, d_i, $i=1,2,...,n$, and the performance of each activity involves a series of cash flow payments and receipts throughout the activity

duration. A terminal value of each activity upon completion can be calculated by compounding the associated cash flows to the end of the activity as follows:

$$C_i = \sum_{t=1}^{d_i} F_{it}\ e^{\alpha(d_i - t)} \qquad [1]$$

where

d_i = duration of activity i (a fixed integer number of periods)

C_i = terminal value of cash flows in activity i at its completion

F_{it} = cash flows for activity i in period t, $t = 1,2,...,d_i$

α = discount rate

A conceptual formulation of the deterministic *unconstrained max-npv problem* can now be formulated as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} q_{f_i}\ C_i \qquad [2]$$

subject to

$$f_i \leq f_j - d_j \qquad \text{for } i = 1,2,...,n \text{ and for each successor } j \text{ of } i \qquad [3]$$

$$f_n \leq T \qquad [4]$$

with C_i and d_i as defined earlier and where

f_i = completion time of activity i (integer variable)

q_t = factor for discounting over t periods to period zero; i.e., $q_t = exp(-\alpha t)$

T = project deadline

The objective (Eq. [2]) is to maximize the net present value of the project. The constraint set given in Eqs. [3] maintains the finish-start precedence relations among the activities. The final constraint (Eq. [4]) limits the project duration to a negotiated project deadline.

The optimal solution procedures for solving the unconstrained 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 optimal solution procedure based on an iterative series of linear approximations and the solution to a transhipment problem over a network model. Apart from an example illustrating the application of the algorithm, he does not report on computational experience. Grinold (1972) also takes an event-oriented view and shows that the problem can be transformed into an equivalent linear program. Again, apart from an example illustrating the computations, no further computational results are given. Icmeli and Erengüç (1996) have adapted the Grinold procedure for AoN networks and have used the procedure for solving unconstrained subproblems in their branch-and-bound procedure for the max-npv problem subject to resource constraints.

Elmaghraby and Herroelen (1990) have developed an optimal 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. In the next section we describe an optimal search procedure for the unconstrained max-npv problem as formulated above in Eqs. [2]-[4].

3. The optimal solution procedure

The algorithm starts by computing the earliest completion time for the activities based on traditional forward pass critical path calculations and determines the corresponding *early tree* in the network which spans all activities (nodes) scheduled at their earliest completion times and which corresponds to a feasible solution with a project duration equal to the critical path length (the arcs of the early tree denote the binding precedence relations). The algorithm then builds the *current tree* by delaying, in reverse order, all nodes with a negative cash flow as much as possible within the early tree; i.e., by linking them to their earliest starting successor. Consider the example AoN network of Figure 1 with a deadline T=20. The current tree, with node *1* as its root, is shown in bold lines. The tree corresponds to a feasible solution with the activity completion times $(f_1,f_2,...,f_9)$ set to (0,4,4,7,8,10,10,12,12).

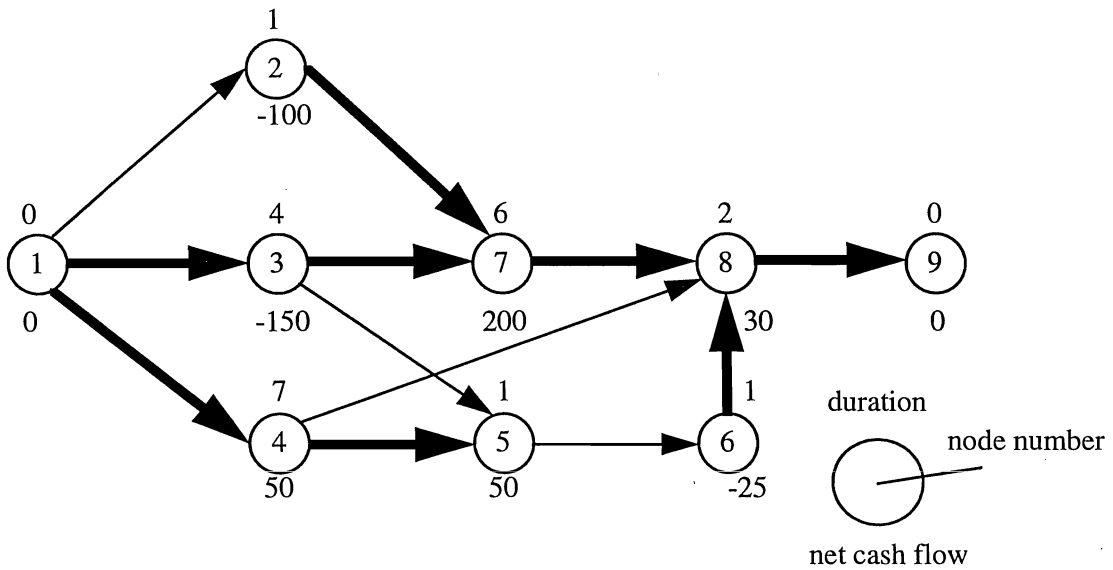

Figure 1. Problem example with current tree

Using the dummy node *1* as the search base, the algorithm will enter a *recursive search* (***recursion(1)*** in the write-up given below) of the current tree to identify partial trees that might be shifted forwards (away from time zero) in order to increase the *npv* of the project. Due to the structure of the recursive search it can never happen that a backward shift (towards time zero) of a partial tree can lead to an increase in the *npv* of the project: any partial tree that is not scheduled at its earliest starting point has a negative *npv* and should be scheduled as late as possible. When a partial tree is the subject of a forward shift, the algorithm computes its minimal displacement interval and updates the current tree. Upon a shift,

the algorithm repeats the recursive search on the current tree associated with the new feasible solution. During the search, it is possible that the current tree disconnects into two parts, one part being shifted forward till it hits the deadline. If this happens, repetitively performing recursion(1) will only optimize the tree connected to node *1*. In order to optimize the second tree we have to perform a similar recursion starting in node *n* (***recursion(n)*** in the write-up). The algorithm stops when no partial trees can be shifted that increase the *npv* of the project.

Using *PT* to denote a partial tree, *DC* to denote the corresponding discounted net cash flow and *A* to denote the set of already examined nodes, the recursive algorithm can be written as follows:

Step 1:

- Compute the *early tree* using standard critical path calculations.
- In reverse order, delay all nodes with a negative cash flow as much as possible within the early tree; i.e., link them to their earliest starting successor.
- Make the resulting tree the *current tree* with activity completion times $(f_1,f_2,...,f_n)$.

Step 2:

- Initialize: $A = \phi$.
- **Do *recursion(1).***

Step 3:

- **If** $n \notin A$

  Do *recursion(n).*

Recursion(newnode)

- Initialize: $PT = \{newnode\}$; $DC = DC_{newnode}$; $A = A \cup \{newnode\}$.
- **Do** for each successor *i* of *newnode* which is not in *A*
  - ***Recursion(i)*** $\rightarrow PT', DC'$
  - **If** $DC' \geq 0$

    Merge: $PT = PT \cup PT'$; $DC = DC + DC'$

    Else

    Update current tree:

    delete *arc(newnode,i)*

    find new arc with minimal displacement

    if an arc exists, add new arc

    if no arc can be found, shift till deadline

    Update completion times of nodes in *PT*

    Repeat step 2 or 3
- **Do** for each predecessor *i* of *newnode* which is not in *A*:
  - ***Recursion(i)*** $\rightarrow PT', DC'$
  - *Merge*: $PT = PT \cup PT'$; $DC = DC + DC'$
- **Return.**

Given the structure of the algorithm, it is clear that only partial trees, consisting of two or more activities, can be the subject of a forward shift during the recursion. When the recursion checks for a predecessor of *newnode*, it is known that the corresponding partial tree must have a negative discounted cash flow and therefore, in this case only a merge is performed.

Consider again the example of Figure 1. Having constructed the current tree with completion times (0,4,4,7,8,10,10,12,12), the algorithm proceeds with Step 2. Using $\alpha = 0.01$, the corresponding solution trace can be sketched as follows:

Step 2: Initialize: $A = \phi$.

Recursion(1)

PT = {1}, DC = 0, A = {1}

Recursion(3): successor node 3

PT = {3}, DC = $-150(e^{-0.01(4)}) = -144.12$, A = {1,3}

Recursion(7): successor node 7

PT = {7}, DC = $200(e^{-0.01(10)}) = 180.97$, A = {1,3,7}

Recursion(8): successor node 8

PT = {8}, DC = $30(e^{-0.01(12)}) = 26.61$, A = {1,3,7,8}

Recursion(9): successor node 9

PT = {9}, DC = 0, A = {1,3,7,8,9}

PT = {8,9}, DC = 0+26.61 = 26.61

Recursion(6): predecessor node 6

PT = {6}, DC = $-25(e^{-0.01(10)}) = -22.62$, A = {1,3,6,7,8,9}

PT = {6,8,9), DC = 26.61-22.62 = 3.99

PT = {6,7,8,9}, DC = 3.99+180.97 = 184.96

Recursion(2): predecessor node 2

PT = {2}, DC = $-100(e^{-0.01(4)}) = -96.08$, A = {1,2,3,6,7,8,9}

PT = {2,6,7,8,9}, DC = 184.96-96.08 = 88.88

PT = {2,3,6,7,8,9}, DC = 88.88-144.12 = *-55.24 < 0*

Delete arc (1,3)

Find next arc (3,5): displacement = $f_5 - d_5 - f_3 = 8-1-4 = 3$

Add arc (3,5) to current tree

Add displacement to completion times of activities in *PT*: $f_2=7$, $f_3=7$, $f_6=13$, $f_7=13$, $f_8=15$, $f_9=15$

Repeat step 2 with the current tree given in Figure 2.

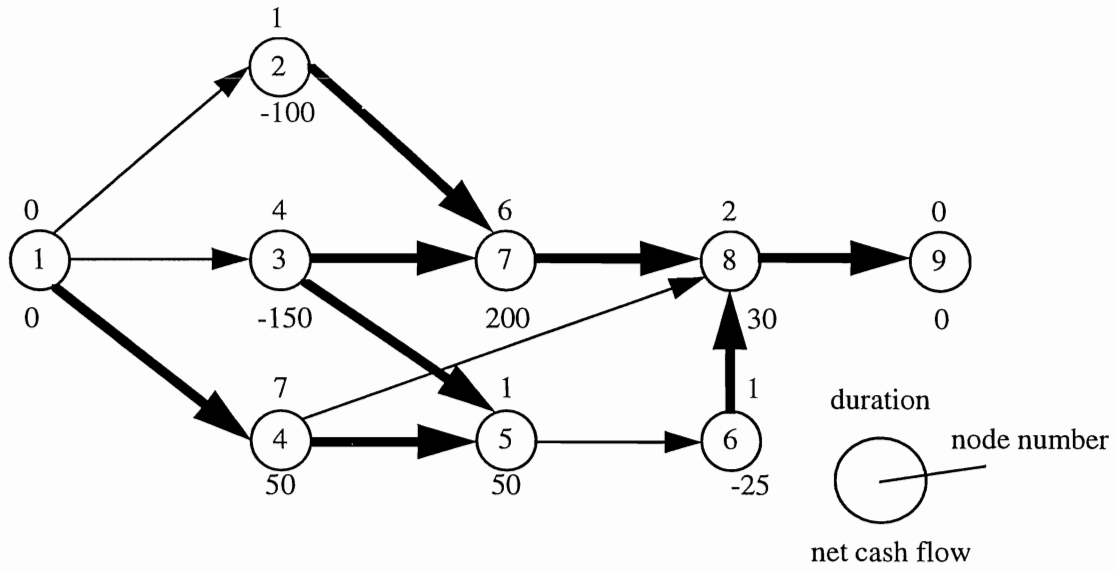

Figure 2. New current tree

The recursion continues with *node 1* as the search base:

Step 2: Initialize: $A = \phi$.

Recursion(1)

PT = {1}, DC = 0, A = {1}

Recursion(4): successor node 4

PT = {4}, DC = $50(e^{-0.01(7)})$ = 46.62, A = {1,4}

Recursion(5): successor node 5

PT = {5}, DC = $50(e^{-0.01(8)})$ = 46.16, A = {1,4,5}

Recursion(3): predecessor node 3

PT = {3}, DC = $-150(e^{-0.01(7)})$ = -139.86, A = {1,3,4,5}

Recursion(7): successor node 7

PT = {7}, DC = $200(e^{-0.01(13)})$ = 175.62, A = {1,3,4,5,7}

Recursion(8): successor node 8

PT = {8}, DC = $30(e^{-0.01(15)})$ = 25.82, A = {1,3,4,5,7,8}

Recursion(9): successor node 9

PT = {9}, DC = 0, A = {1,3,4,5,7,8,9}

PT = {8,9}, DC = 0+25.82 = 25.82

Recursion(6): predecessor node 6

PT = {6}, DC = $-25(e^{-0.01(13)})$ = -21.95, A = {1,3,4,5,6,7,8}

PT = {6,8,9}, DC = 25.82-21.95 = 3.87

PT = {6,7,8,9}, DC = 175.62+3.87 = 179.49

Recursion(2): predecessor node 2

PT = {2}, DC = $-100(e^{-0.01(7)})$ = -93.24, A = {1,2,3,4,5,7,8}

PT = {2,6,7,8,9}, DC = 179.49-93.24 = 86.25

PT = {2,3,6,7,8,9}, DC = 86.25-139.86 = -53.61

PT = {2,3,5,6,7,8,9}, DC = 46.16-53.61 = *-7.45 < 0*

Delete arc (4,5)

No new arc found

Shift partial tree PT = {2,3,5,6,7,8,9} till deadline T=20

Update completion times of activities in *PT*: f_2=12, f_3=12, f_5=13, f_6=18, f_7=18, f_8=20, f_9=20

Repeat step 2 with the disconnected current trees given in Figure 3.

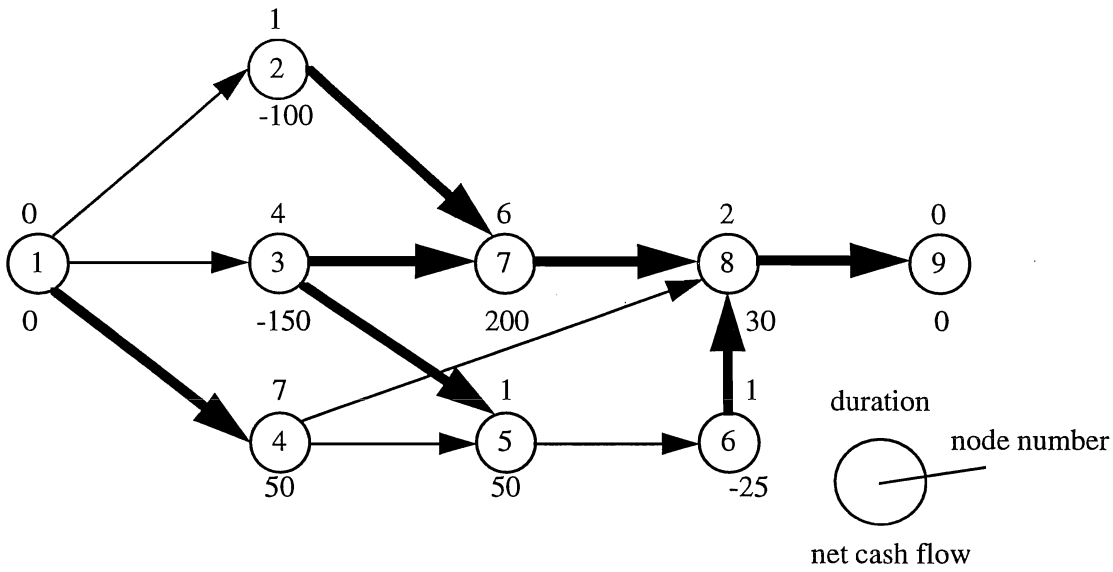

Figure 3. Disconnected current trees representing the optimal solution

The search continues on the current tree containing node 1.

Step 2: Initialize: $A = \phi$.

Recursion(1)

PT = {1}, DC = 0, A = {1}

Recursion(4): successor node 4

PT = {4}, DC = $50(e^{-0.01(7)})$ = 46.62, A = {1,4}

PT = {1,4}, DC = 0+46.62 = 46.62

Return

The recursion continues with step 3.

Step 3: Node n=9 does not belong to A= {1,4}

Recursion(9)

PT = {9}, DC = 0, A = {9}

Recursion(8): predecessor node 8

PT = {8}, DC = $30(e^{-0.01(20)})$ = 24.56, A = {8,9}

Recursion(6): predecessor node 6

PT = {6}, DC = $-25(e^{-0.01(18)})$= -20.88, A = {6,8,9}

PT = {6,8}, DC = 24.56-20.88 = 3.68

Recursion(7): predecessor node 7

PT = {7}, DC = $200(e^{-0.01(18)})$= 167.05, A = {6,7,8,9}

Recursion(2): predecessor node 2

PT = {2}, DC = $-100(e^{-0.01(12)})$= -88.69, A = {2,6,7,8,9}

PT = {2,7}, DC = 167.05-88.69 = 78.36

Recursion(3): predecessor node 3

PT = {3}, DC = $-150(e^{-0.01(12)})$= -133.04, A = {2,3,6,7,8,9}

Recursion(5): successor node 5

PT = {5}, DC = $50(e^{-0.01(13)})$= 43.90, A = {2,3,5,6,7,8,9}

PT = {3,5}, DC = 43.90-133.04 = -89.14

PT = {2,3,5,7}, DC = 78.36-89.14 = -10.78

PT = {2,3,5,6,7,8}, DC = 3.68-10.78 = -7.10

PT = {2,3,5,6,7,8,9}, DC = 0-7.10 = -7.10

Return

The optimal solution is represented by the disconnected current trees represented in Figure 3: f_1=0, f_2=12, f_3=12, f_4=7, f_5=13, f_6=18, f_7=18, f_8=20, f_9=20 with an *npv* equal to the sum of the *npv's* of the two resulting partial trees: npv = 46.62-7.10 = 39.52.

4. Computational results

The recursive search algorithm has been programmed in C using the Visual C/C++ compiler V2.1 by Microsoft® for a WindowsNT platform on a Digital Pentium 60 Mhz Venturis machine. As mentioned earlier, the previously developed optimal procedures for solving the deterministic unconstrained max-npv problem take an event-oriented view and aimed at AoA networks. Inspired by Icmeli and Erengüç (1995), we have adapted the procedure developed by Grinold (1972), considered to be one of the most efficient, for AoN networks and we have coded the resulting algorithm on the same platform. In doing so, we have diverted from Grinold in using the current tree (i.e., the early tree with all negative cash flow nodes scheduled as late as possible) as our starting solution. This considerably reduces the number of equation solving iterations.

Both algorithms have been tested on the same two data sets. The first data set is adapted from the well-known 110 networks assembled by Patterson (1984) for the resource-constrained project scheduling problem. Retaining the instances with less than 32 activities (i.e., the number of activities ranging from 7 to 27) we obtain 98 AoN networks for which the resource data have been omitted and for which cash flows have been generated from the uniform distribution within the range +/- 510, using the random number generator provided by the C/C++ compiler. The second set consists of 1980 networks with activity durations uniformly distributed between 0 and 10 and the number of activities ranging from 10 to 30, taken from the 6000 networks generated by De Reyck and Herroelen (1995) as instances for the assembly line balancing problem (ALB). Cash flows were generated from the same uniform distribution. For both data sets, the due dates were set to 9999.

The CPU times reported below for solving a problem instance actually relate to the total time required for solving 1000 replications of the same problem. The reasons for doing this are twofold. First, the required solution time for each problem was very small. Second, branch-and-bound procedures for the

resource-constrained max-npv problem may base their lower bound calculations at a node in the search tree on the optimal solution of the corresponding unconstrained problem (Icmeli and Erengüç 1996). As such, the total time needed by an algorithm to solve a large number of unconstrained max-npv problems reflects its potential use in a branch-and-bound scheme for the constrained problem.

4.1 The adapted Patterson problem set

The results obtained by the search algorithm and the adapted Grinold procedure on the 98 problems adapted from the Patterson problem set are given in Table I. The instances are grouped in three classes: networks with less than 22 activities, networks with 22 or 23 activities and networks with 27 activities. For the problems in each class we report the average CPU time (in seconds) and the corresponding standard deviation for solving 1000 replications of each problem.

Table I. Results on the reduced Patterson problem set

Number of activities	**Number of networks (1000 replications)**	**CPU time* Grinold**		**CPU time* Search**	
		Mean	*Standard deviation*	*Mean*	*Standard deviation*
< 22	8	0.276	0.371	0.125	0.095
22, 23	47	1.091	0.599	0.423	0.155
27	43	1.274	0.506	0.501	0.158
All	*98*	*1.105*	*0.601*	*0.433*	*0.182*

* Time measured in seconds on a Digital Pentium 60 Mhz Venturis machine

It can be seen from this table that the average solution time required by both algorithms to solve 1000 replications of a problem instance increases with the size (i.e., the number of nodes) of the corresponding network. The search algorithm outperforms the adapted Grinold procedure in that it is on the average approximately 2.5 times faster while the standard deviation of the CPU time is much smaller.

4.2 The ALB problem set

The results obtained on the 1980 ALB instances are given in Table II. Again the computational requirements for both algorithms increase with problem size. The search algorithm consistently outperforms the adapted Grinold procedure. It requires an average of only 0.420 seconds to solve 1000 replications of each problem whereas Grinold requires an average of 1.088 seconds, resulting in an average comparison factor of 2.59. Again, the search algorithm obtains its results with a much smaller standard deviation of the CPU time. The results obtained on this extensive problem set confirm the efficiency of the search algorithm in solving the unconstrained max-npv problem. Moreover, they hold a strong indication of the potential of the search procedure for computing lower bounds in a branch-and-bound procedure for the resource-constrained max-npv problem.

Inspired by the experiments performed by De Reyck and Herroelen (1996) for testing the potential of the complexity index (*CI*) as an explaining factor for the underlying network complexity, Table III shows a breakdown of each network class into subclasses corresponding to various ranges of values for the *CI*. Empty cells refer to the fact that the problem class contains no instances with the corresponding values for the complexity index. The complexity index (*CI*) denotes the minimum number of node reductions which are necessary to reduce a network to a single activity (Bein et al. 1992).

Obviously, the same increase in computational requirements with problem size can be observed for both procedures. Moreover, for the same number of activities, the CPU times required by both procedures, clearly increase with increasing values of the complexity index.

Table II. Results obtained on the ALB problem set

Number of activities	**CPU time* Grinold**		**CPU time* Search**	
	Mean	*Standard deviation*	*Mean*	*Standard deviation*
10	0.252	0.126	0.135	0.045
15	0.568	0.303	0.259	0.105
20	0.991	0.567	0.400	0.190
25	1.608	0.896	0.589	0.285
30	2.022	1.096	0.719	0.330
All	***1.088***	***0.954***	***0.420***	***0.305***

* Time measured in seconds on Digital Pentium 60 Mhz Venturis machine

Table III. Computational requirements versus the complexity index

# activities	10		15		20		25		30	
CI	G*	S**	G	*S*	G	*S*	G	*S*	G	*S*
0..2	0.244	*0.131*	0.454	*0.213*	0.725	*0.307*				
3..8	0.271	*0.146*	0.577	*0.262*	0.792	*0.333*	1.114	*0.436*	1.430	*0.532*
9..13					1.208	*0.473*	1.749	*0.612*	1.823	*0.677*
14..19							2.101	*0.780*	2.495	*0.855*

* Mean CPU time obtained by the adapted Grinold procedure
** Mean CPU time obtained using the search procedure

5. Conclusions

In this paper an optimal search procedure was described for the deterministic unconstrained max-npv problem in AoN networks. Activities have a known deterministic duration and are subject to zero-lag finish-start precedence constraints. Cash flows are assumed to be known in both their amount and timing. Progress payments and cash outflows are made upon the completion of certain activities. The project is to be scheduled subject to a fixed deadline in the absence of resource constraints. The optimal procedure performs a recursive search on partial tree structures exploiting the fundamental logic 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 procedure has been coded in Visual C++ for use on a personal computer. The procedure was validated against Grinold's procedure, adapted for AoN networks. Extensive computational tests obtained on a Digital Pentium 60 Mhz Venturis machine on two data sets (98 test problems adapted from the 110 Patterson problem set and 1980 networks adapted from the De Reyck & Herroelen set of ALB test problems) reveal that the recursive search algorithm is very efficient. It finds the optimal solution for 1000 problem replications in an average of 0.433 seconds for the Patterson set and 0.420 seconds for the ALB problem set. It outperforms Grinold's procedure in that it is on the average 2.5 times faster on the Patterson set and 2.59 times faster on the ALB set at a much smaller CPU time variance. The encouraging computational results confirm that the unconstrained max-npv problem can be solved optimally at low

computational cost. They hold the promise that the efficient recursive search procedure may be effectively used in branch-and-bound schemes for solving the max-npv problem subject to resource constraints.

References

Bein, W.W., J. Kamburowski and M.F.M. Stallmann, "Optimal Reduction of Two-terminal Directed Acyclic Graphs", *SIAM Journal of Computing*, 21(1992), 1112-1129.

De Reyck, B. & W. Herroelen, "Assembly Line Balancing by Resource-constrained Project Scheduling Techniques - A Critical Appraisal", Research Report N° 9505, Department of Applied Economics, K.U.Leuven, 1995a.

De Reyck, B. & W. Herroelen, "On the Use of the Complexity Index as a Measure of Complexity in Activity Networks", *European Journal of Operational Research*, 1996, to appear.

Elmaghraby, S.E. and W.S. Herroelen, "The Scheduling of Activities to Maximize the Net Present Value of Projects", *European Journal of Operational Research*, 49(1990), 35-49.

Grinold, R.C., "The Payment Scheduling Problem", *Naval Research Logistics Quarterly*, 19(1972), 123-136.

Herroelen, W.S. and E. Gallens, "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(1993), 274-277.

Herroelen, W.S., P. Van Dommelen and E.L. Demeulemeester, "Project Network Models with Discounted Cash Flows - A Guided Tour through Recent Developments", Research Report N° 9528, Department of Applied Economics, K.U.Leuven, 1995.

Icmeli, O. and S.S. Erengüç, "A Branch and Bound Procedure for the Resource Constrained Project Scheduling Problem with Discounted Cash Flows", *Management Science*, 1996, to appear.

Patterson, J.H., "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem", *Management Science*, 30(1984), 854-867.

Russell, A.H., "Cash Flows in Networks", *Management Science*, 16(1970), 357-373.