Journal of the Operational Research Society (1998) 49, 1153–1163
http://www.stockton-press.co.uk/jor

New computational results on the discrete time/cost trade-off problem in project networks

E Demeulemeester[1], B De Reyck[2], B Foubert[3], W Herroelen[1] and M Vanhoucke[1]

[1]*Katholieke Universiteit, Belgium,*[2]*Erasmus University, The Netherlands and* [3]*University of Antwerp, Belgium*

We describe a new exact procedure for the discrete time/cost trade-off problem in deterministic activity-on-the-arc networks of the CPM type, where the duration of each activity is a discrete, nonincreasing function of the amount of a single resource (money) committed to it. The objective is to construct the complete and efficient time/cost profile over the set of feasible project durations. The procedure uses a horizon-varying approach based on the iterative optimal solution of the problem of minimising the sum of the resource use over all activities subject to the activity precedence constraints and a project deadline. This optimal solution is derived using a branch-and-bound procedure which computes lower bounds by making convex piecewise linear underestimations of the discrete time/cost trade-off curves of the activities to be used as input for an adapted version of the Fulkerson labelling algorithm for the linear time/cost trade-off problem. Branching involves the selection of an activity in order to partition its set of execution modes into two subsets which are used to derive improved convex piecewise linear underestimations. The procedure has been programmed in Visual C++ under Windows NT and has been validated using a factorial experiment on a large set of randomly generated problem instances.

Keywords: project management; CPM; time/cost trade-off; branch-and-bound

Introduction

The specific problem addressed in this paper is the discrete time/cost trade-off problem in project networks of the CPM type. The specification of a project is assumed to be given in activity-on-the-arc (AoA) notation by a directed acyclic graph (dag) $D = (N, A)$ in which N is the set of nodes, representing network 'events', and A is the set of arcs, representing network 'activities'. We assume, without loss of generality, that there is a single start node 1 and a single terminal node n, $n = |N|$. The duration y_a of activity $a \in A$ is a discrete, nonincreasing function $g_a(x_a)$ of the amount of a single resource (money) allocated to it; namely, $y_a = g_a(x_a)$. The pair y_a, x_a shall be referred to as a 'mode', and shall be written as: (y_a, x_a). Therefore an activity that assumes four different durations according to four possible resource allocations to it shall be said to possess four modes.

The early contributions to the basic time/cost trade-off problem in CPM networks[1] assumed ample resource availability and hence did not explicitly take resource decisions into account. A direct activity cost function was used instead, representing the direct activity costs as a function of activity duration. Activity durations are bounded from below by the crash duration (corresponding to a maximum allocation of resources) and bounded from above by the normal duration (corresponding to the most efficient resource allocation). Essentially, the project costs correspond to a requirement for a nonrenewable resource, the total requirement of which is to be minimised. This corresponds to minimising the (required) availability of the resource.

Three possible objective functions have been studied in the literature.[2] For the first objective function (referred to as problem 1, $T|cpm, disc, mu|C_{\max}$ in the classification scheme of Herroelen *et al*[3]) a limit R is specified on the total availability of a single nonrenewable resource type. The problem is then to decide on the vector of activity durations $(y_1, \ldots, y_m)$, $m = |A|$, that completes the project as early as possible under the limited availability of the single nonrenewable resource type. A second objective function (referred to as problem 1, $T|cpm, \delta_n, disc, mu|av$) reverses this problem formulation: now there is a limit δ_n on the project length and we try to minimise the sum of the resource use over all activities. For the third objective function (referred to as problem 1, $T|cpm, disc, mu|curve$) the complete time/cost trade-off function for the total project is to be computed, that is, all the efficient points (T, R) such that with a resource limit R a project length T can be obtained and such that no other point (T', R') exists for which both T' and R' are smaller than or equal to T and R.

When the activity cost functions are linear, the problem is denoted as 1, $T|cpm, \delta_n, lin, mu|av$ in the classification scheme of Herroelen *et al*.[3] The problem can be solved optimally by the well-known Fulkerson maximum flow algorithm.[4] This method finds a flow augmenting path in a network and increases the flow value along this path until at least one such path remains in the network. Several other

Correspondence: Dr W Herroelen, Department of Applied Economics, Katholieke Universiteit Leuven, Naamsestraat 69, B-3000, Leuven, Belgium.
E-mail: willy.herroelen@econ.kuleuven.ac.be

efficient algorithms have been proposed, also for other types of continuous time/cost functions. For an extensive review, we refer the reader to Ahuja *et al.*[5] While the problem has been widely studied under the assumption of continuous time/cost relationships,[3] the literature on the discrete case has been rather sparse. De *et al*[6] have shown that the discrete time/cost trade-off problem is NP-hard under the three objectives mentioned above. De *et al*[2] offer an excellent review of the literature. Demeulemeester *et al*[7] reported on promising computational experience with two exact procedures. The first algorithm is based on a procedure for finding the minimal number of reductions necessary to transform a general network to a series-parallel network. The second algorithm minimises the computational effort in enumerating alternative modes through a branch-and-bound search tree.

The objective of this paper is to present and validate a new optimal procedure for problem 1, $T|cpm, disc, mu|curve$. The paper is organised as follows: In Section 2 we clarify the solution methodology. Section 3 provides an illustrative problem example. Computational experience is reported in Section 4. Overall conclusions are offered in the last section.

An exact procedure

The solution procedure presented in this paper provides an optimal solution to problem 1, $T|cpm, disc, mu|curve$ using a horizon-varying approach which involves the iterative optimal solution of problem 1, $T|cpm, \delta_n, disc, mu|av$ (minimise the sum of the resource use over all activities subject to a project deadline) over the feasible project durations in the interval bounded from below by $\underline{t}_n$ (the project duration obtained with the activity crash modes) and from above by $\bar{t}_n$ (the project duration obtained under normal conditions).

The branch-and-bound algorithm

Problem 1, $T|cpm, \delta_n, disc, mu|av$ is solved using a branch-and-bound algorithm which is based on the following logic. The algorithm starts by computing for each activity a convex piecewise linear underestimation of its discrete time/cost trade-off curve as shown in bold lines in Figure 1. An initial lower bound is obtained by solving the resulting time/cost trade-off problem using a variant of the well-known Fulkerson labelling algorithm.[4] The algorithm, which is described below, yields for each activity a mode (x°, y°).

For each activity and associated mode (x°, y°), a *vertical distance* vd_{ij}, is computed which measures the quality of the convex underestimation. Two rules may be used to perform this computation. Rule 1 computes vd_{ij} as the minimum of the distances $vd1$ and $vd2$. $vd1$ is the distance between y° and the cost of the nearest mode to the left of x° on the convex piecewise linear underestimation (cost y_2 in Figure 1). $vd2$ is the distance between y° and the cost of the nearest mode to the right of x° on the convex piecewise linear underestimation (cost y_4 in Figure 1). Rule 2 computes the vertical distance between y° and the linear interpolation between the nearest mode to the left of x° and the nearest mode to the right of x° (distance $vd3$ in Figure 1). Notice that $vd3$ is equal to zero when the convex piecewise linear underestimation connects the mode to the left of x° and the

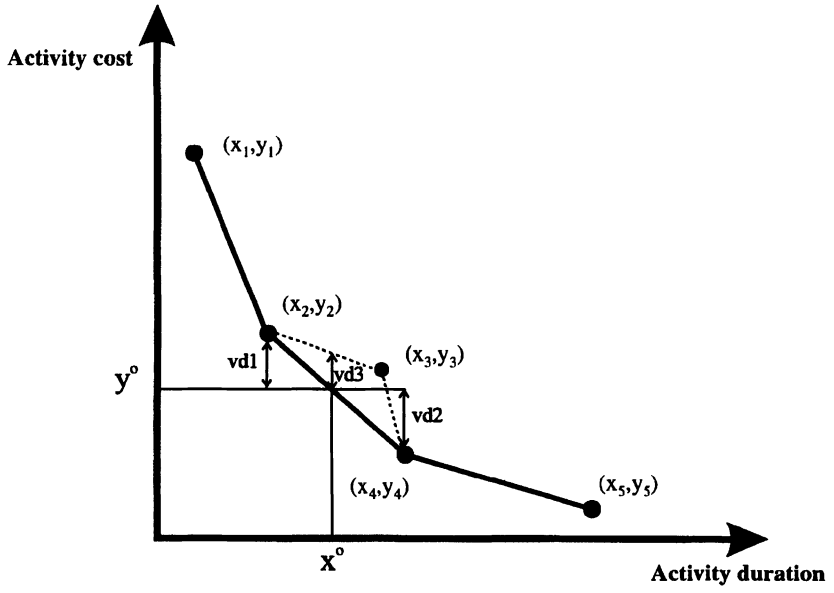

Figure 1 Convex piecewise linear underestimation of an activity's discrete time/cost trade-off profile.

mode to the right of x°. Therefore, Rule 2 uses Rule 1 as a tie-breaker.

Branching is done by identifying the activity with the largest vertical distance and partitioning its set of modes into two subsets. The first subset, set_1, consists of the set of modes with a duration greater than x° (modes (x_3, y_3), (x_4, y_4) and (x_5, y_5) in Figure 1). The second subset, set_2, consists of the set of modes with a duration smaller than or equal to x° (modes (x_1, y_1) and (x_2, y_2) in Figure 1). These subsets are used to obtain two new convex piecewise linear underestimations for the activity. The first descendant node in the search tree replaces the current underestimation for the activity by the one provided by set_1 (Figure 2b), the second descendant node replaces the current underestimation by the one provided by set_2 (Figure 2a). Notice that the new underestimations may provide a closer fit to the original time/cost trade-off profile. For the example, this is the case for set_1 (Figure 2b). The solution of the two corresponding problems using the adapted Fulkerson algorithm yields the corresponding new lower bounds. Branching continues from the node with the smallest lower bound (arbitrary tie-break).

Backtracking occurs when the lower bound exceeds (or equals) the cost of an earlier found schedule or when no feasible solution can be found for the modified convex piecewise linear underestimation using the modified Fulkerson algorithm. The procedure stops when backtracking leads to the source node of the search tree.

Computing the lower bound

In the original Fulkerson labelling algorithm[4] for solving problem 1, $T|cpm, \delta_n, lin, mu|av$, the flow capacity of an arc is associated with the negative slope of the corresponding activity's (linear) direct cost curve. The algorithm is adapted for convex piecewise trade-off curves as follows. Each time the duration of an activity (arc) is reduced to a value which coincides with a breakpoint in the piecewise linear underestimating cost curve the arc's capacity is updated. The way this is done is best illustrated using the time/cost trade-off curve presented in Figure 3.

The breakpoints of the curve correspond to the modes (20, 5), (16, 7), (13, 10) and (8, 17). Each time the activity's duration switches between the intervals [8, 8],]8, 13],]13, 16] and]16, 20], the corresponding arc capacity is updated. At the start of the algorithm, the duration of the activity is initialised to its normal value (20), while the arc capacity is set equal to 0.5, the negative value of the slope of the line segment connecting points (20, 5) and (16, 7). Suppose the activity's duration is reduced to 16. It is clear that the marginal cost value of 0.5 is no longer valid as a further reduction in duration can only be obtained at a marginal cost of 1, the (negative) value of the slope of the line segment connecting points (16, 7) and (13, 10). Therefore, the (residual) capacity is increased by 0.5 (the difference between 1 and 0.5). Suppose the activity's duration is reduced to 8. Further reduction is impossible: the marginal cost is ∞, the negative slope of the line segment connecting points (8, 17) and (8, ∞). The arc capacity is set to ∞, in a similar way, an increase in an activity duration would lead to a decrease in the corresponding (residual) arc capacity. Assume the duration of the activity is extended from 13 (where the curve shows a breakpoint) to 15. The (residual) capacity is now reduced by 0.4, the difference between 1.4 (the negative slope of the line segment connecting points (13, 10) and (8, 17)) and 1 (the negative slope of the line segment connecting points (16, 7) and (13, 10)).

The coded version of the adapted Fulkerson algorithm exhibits worst-case complexity $O(n^2m)$. Despite this relatively high worst-case complexity, in practice the

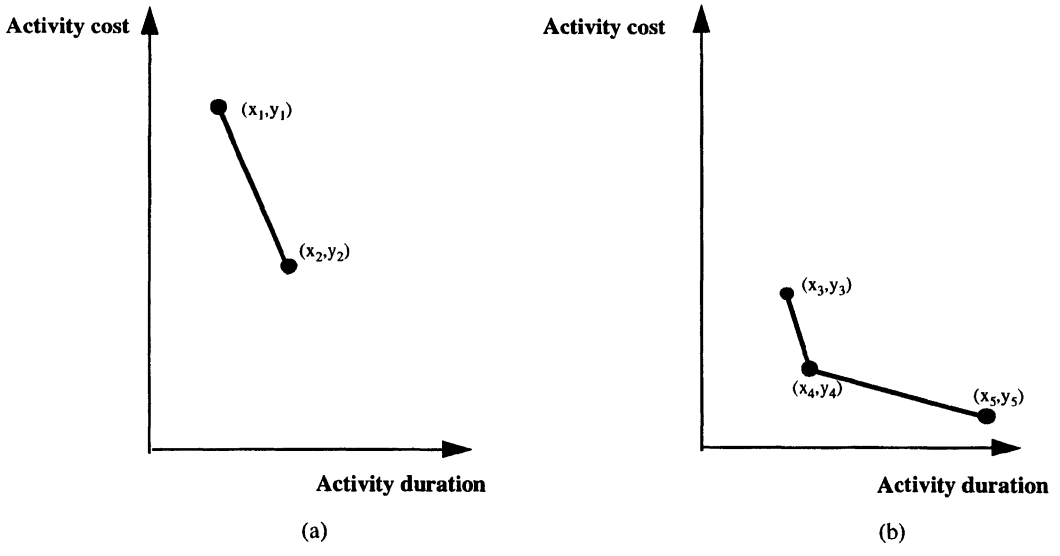

Figure 2 Two convex piecewise linear underestimations for the selected activity.

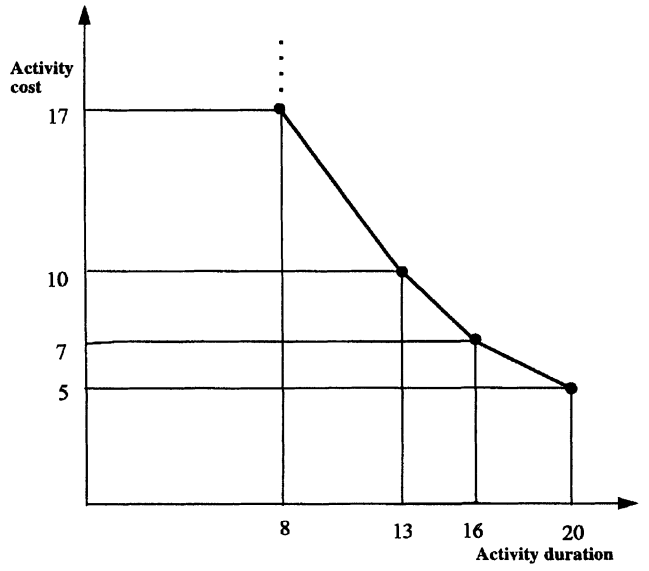

Figure 3 An underestimating convex piecewise linear time/cost trade-off curve.

computational results reported below reveal very small required computation times.

The horizon-varying algorithm

The horizon-varying algorithm computes a convex piecewise linear underestimation for the time/cost trade-off profile using a set of durations MS_{ij}. The negative slopes of the line segments of the convex underestimation are saved in CC_{ij}, the convex set of marginal costs of activity (i, j). If activity (i, j) only has one mode, $CC_{ij} = \{\}$. The activity durations which correspond to the modes lying on the convex underestimating curve are saved in the set CD_{ij}, the convex set of durations of activity (i, j).

Step 0. Deadline computation

- Compute $\bar{t}_n$, the critical path length with every activity (i, j) at its normal duration.
- Compute $\underline{t}_n$, the critical path length with every activity (i, j) at its crash duration.
- Set the current project deadline $T = \bar{t}_n$.

Step 1. Initialisation

- Let $ub(T) = \infty$, the upper bound of the project cost for deadline T.
- Set p, the level of the search tree, equal to 0.
- Compute for every activity (i, j) the convex piecewise linear underestimation with the set of durations $MS_{ij} = M_{ij}, M_{ij}$ being the original set of possible durations d_{ij} for activity (i, j).
- Run the adapted Fulkerson labelling algorithm to compute a lower bound lb and the corresponding activity durations w_{ij}.

Step 2. Identify the activity with the maximal vertical distance

- Compute for each activity (i, j) its vertical distance vd_{ij}.
- Compute the maximal vertical distance $vd_{\max} = \max_{(i,j)\in A}\{vd_{ij}\}$.
- If $vd_{\max} = 0$, the schedule is complete and feasible. Update $ub(T) = lb$ and go to step 4.
- Update the level of the branch-and-bound tree: $p = p + 1$.
- Store the activity (u, v) with $vd_{uv} = vd_{\max}$ at level p (ties are broken arbitrarily). Store the corresponding sets MS_{uv}, CC_{uv} and CD_{uv}.

Step 3. Separate and branch

- Generate two descendant nodes in the search tree. For the first node, define $set_1 = \{d_{uv} \in MS_{uv} | d_{uv} > w_{uv}\}$ and compute the convex piecewise linear underestimation with $MS_{uv} = set_1$. Store durations w_{ij} and lower bound lb_1. For the second mode, define $set_2 = \{d_{uv} \in MS_{uv} | d_{uv} \leqslant w_{uv}\}$ and compute the convex piecewise linear underestimation with $MS_{uv} = set_2$. Store durations w_{ij} and lower bound lb_2.
- Select the node with the smallest lower bound $lb = \min(lb_1, lb_2)$ for branching. If $lb \geqslant ub(T)$, go to step 4.
- Store the information for the remaining node r. Go to step 2.

Step 4. Backtracking

- If the branching level $p = 0$, then go to step 5.
- If both nodes at level p have been evaluated, set $p = p - 1$ and repeat step 4.
- Evaluate the remaining node r at this level: if $lb_r \geqslant ub(T)$, set $p = p - 1$ and repeat step 4.
- Go to step 2.

Step 5. Update project deadline

- Set $T = T - 1$. If $T < \underline{t}_n$, stop; else, go to step 1.

In the worst case and for a given deadline, the algorithm generates $2\prod_{i=1}^{m} k_i - 1$ nodes in the search tree (k_i being the number of modes of activity i) yielding a worst-case complexity of $O(k^m)$, where k denotes the maximum number of modes over each of the m activities. For each node in the search tree, a lower bound is computed using the adapted Fulkerson algorithm, which is of worst case complexity $O(n^2 m)$.

An illustrative example

Consider the AoA network shown in Figure 4. The cost/duration profiles and the first convex piecewise linear underestimation for activities (1,2), (1,3), (2,4) and (3,4) are shown in Figure 5a through 5d. The original sets of activity durations are $M_{12} = \{11, 5\}$, $M_{13} = \{10, 6, 3, 1\}$,

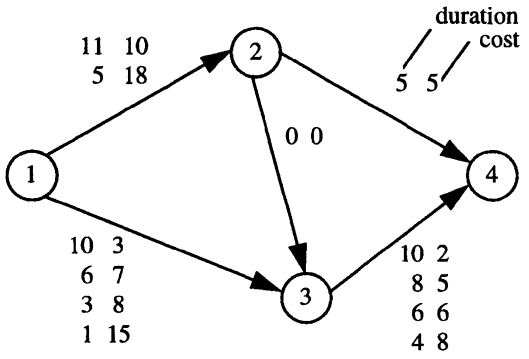

Figure 4 Project network example.

$M_{23} = \{0\}$, $M_{24} = \{5\}$ and $M_{34} = \{10, 8, 6, 4\}$. We illustrate the branch-and-bound procedure for a deadline $T = 14$. This will yield one point on the project's time/cost trade-off profile. The resulting branch-and-bound search tree is shown in Figure 6.

Step 1. Initialize $ub(T) = \infty$ and $p = 0$. Compute the convex piecewise linear under estimation with $MS_{ij} = M_{ij}$. This yields the convex set of durations $CD_{12} = \{11, 5\}$, $CD_{13} = \{10, 3, 1\}$, $CD_{23} = \{0\}$, $CD_{24} = \{5\}$ and $CD_{34} =$

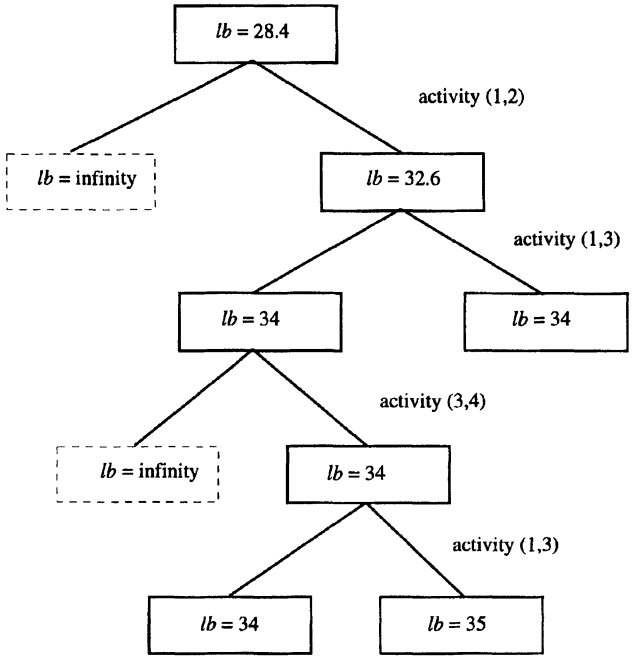

Figure 6 Branch-and-bound search tree for the problem example.

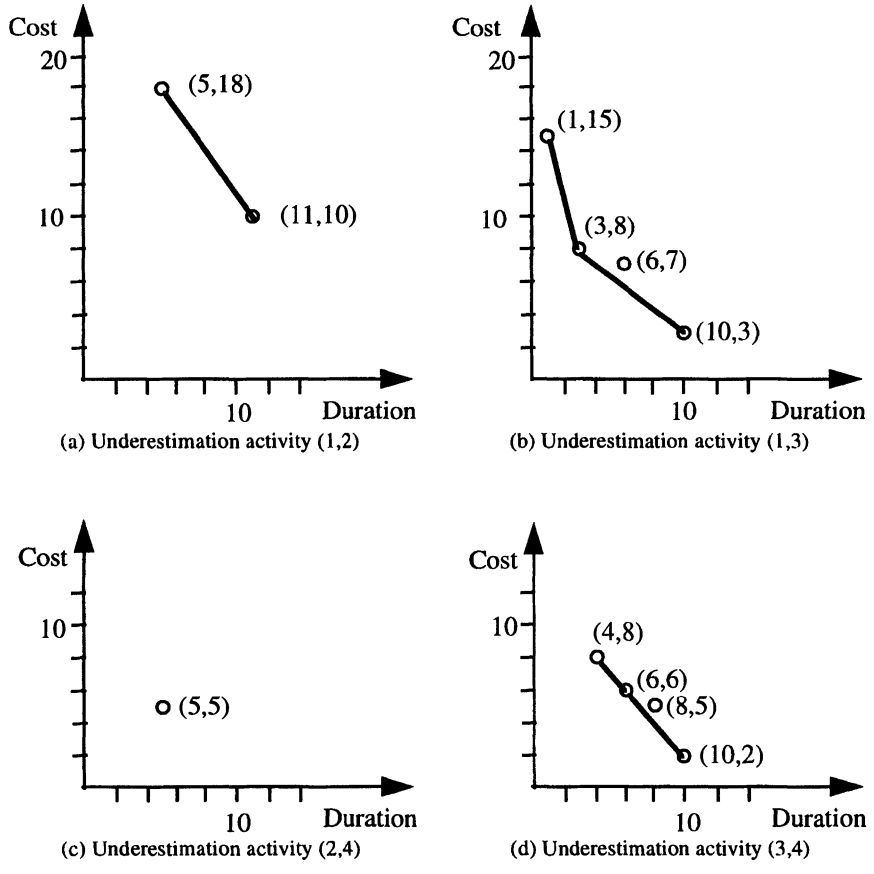

Figure 5 Underestimations for the activities of the example network.

$\{10, 6, 4\}$. The convex set of marginal costs is computed as $CC_{12} = \{1.33\}$, $CC_{13} = \{0.71, 3.5\}$, $CC_{23} = \{\ \}$, $CC_{24} = \{\ \}$, and $CC_{34} = \{1, 1\}$. The adapted Fulkerson algorithm yields a lower bound $lb = 28.4$ and corresponding activity durations $w_{12} = 9$, $w_{13} = 9$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 5$.

Step 2. Let us assume that the vertical distances are computed according to Rule 1: $vd_{12} = 2.67$, $vd_{13} = 0.71$, $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 1$. As an illustration, vd_{13} is computed as follows. The duration $w_{13} = 9$ corresponds to a cost on the piecewise linear underestimating curve of $3 + 0.71 \times (10 - 9) = 3.71$. Then $vd_{13} = \min\{8 - 3.71; 3.71 - 3\} = 0.71$. The maximal distance is $vd_{12} = 2.67$. Activity $(1, 2)$ has the largest vertical distance. Store the corresponding information: $(u, v) = (1, 2)$, $MS_{12} = \{11, 5\}$, $CC_{12} = \{1.33\}$, $CD_{12} = \{11, 5\}$. Update the branching level: $p = 1$.

It is interesting to observe that Rule 2 would have computed the vertical distances as follows: $vd_{12} = 0$, $vd_{13} = 0.29$ (that is, $4 - 3.71$), $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 0$. As a result, activity $(1, 3)$ would be identified as the activity with the maximal vertical distance.

Step 3. Generate the two descendant nodes at level $p = 1$ of the search tree. $set_1 = \{11\}$ and $set_2 = \{5\}$. Compute the underestimation with $MS_{12} = set_1 = \{11\}$, $CC_{12} = \{\ \}$ and $CD_{12} = \{11\}$. The adapted Fulkerson procedure yields $lb_1 = \infty$. Compute the underestimation with $MS_{12} = set_2 = \{5\}$, $CC_{12} = \{\ \}$ and $CD_{12} = \{5\}$. The adapted Fulkerson procedure yields $lb_2 = 32.6$ and $w_{12} = 5$, $w_{13} = 5$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 9$. Select the second node for branching.

Step 2. Compute the vertical distances: $vd_{12} = 0$, $vd_{13} = 1.43$, $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 1$. The maximal vertical distance is 1.43 for activity $(1, 3)$. Increase the branching level: $p = 2$. Store the corresponding information: $(u, v) = (1, 3)$, $MS_{13} = \{10, 6, 3, 1\}$, $CC_{13} = \{0.71, 3.5\}$ and $CD_{13} = \{10, 3, 1\}$.

Step 3. Generate the two descendant nodes at level 2 of the search tree: $set_1 = \{10, 6\}$ and $set_2 = \{3, 1\}$. Compute the underestimation with $MS_{13} = \{10, 6\}$, $CC_{13} = \{1\}$ and $CD_{13} = \{10, 6\}$. The adapted Fulkerson procedure yields $lb_1 = 34$ and $w_{12} = 5$, $w_{13} = 6$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 8$. Compute the underestimation for $MS_{13} = \{3, 1\}$, $CC_{13} = \{3.5\}$ and $CD_{13} = \{3, 1\}$ and run the Fulkerson procedure: $lb_2 = 34$ and $w_{12} = 5$, $w_{13} = 3$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 9$. Select the first node for branching with $lb = 34$ (arbitrary tie-break).

Step 2. Compute the vertical distances: $vd_{12} = 0$, $vd_{13} = 0$, $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 2$. Activity $(3, 4)$ is the current activity with the maximal vertical distance. Store the corresponding information: $(u, v) = (3, 4)$, $MS_{34} = \{10, 8, 6, 4\}$, $CC_{34} = \{1, 1\}$ and $CD_{34} = \{10, 6, 4\}$. Update the level of the search tree: $p = 3$.

Step 3. Generate the two descendant nodes at level $p = 3$: $set_1 = \{10\}$ and $set_2 = \{8, 6, 4\}$. Compute the underestimation for $MS_{34} = \{10\}$, $CC_{34} = \{\ \}$ and $CD_{34} = \{10\}$ and use the Fulkerson algorithm: $lb_1 = \infty$. Compute the underestimation for $MS_{34} = \{8, 6, 4\}$, $CC_{34} = \{0.5, 1\}$ and $CD_{34} = \{8, 6, 4\}$ and use the Fulkerson routine to find $lb_2 = 34$ and $w_{12} = 5$, $w_{13} = 8$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 6$. Select the second node for branching with $lb = 34$.

Step 2. Compute the vertical distances: $vd_{12} = 0$, $vd_{13} = 2$, $vd_{23} = 0$, $vd_{24} = 0$, $vd_{34} = 0$. The maximal vertical distance is 2 for activity $(1, 3)$. Increase the branching level: $p = 4$. Store the corresponding information: $MS_{13} = \{10, 6\}$, $CC_{13} = \{1\}$ and $CD_{13} = \{10, 6\}$.

Step 3. Generate the two descendant nodes at level $p = 4 : set_1 = \{10\}$ and $set_2 = \{6\}$. Underestimate for $MS_{13} = \{10\}$: $CC_{13} = \{\ \}$ and $CD_{13} = \{10\}$ and use the Fulkerson procedure to find: $lb_1 = 34$ and $w_{12} = 5$, $w_{13} = 10$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 4$. Underestimate for $MS_{13} = \{6\}$, $CC_{13} = \{\ \}$ and $CD_{13} = \{6\}$ and run the Fulkerson routine to find: $lb_2 = 35$ and $w_{12} = 5$, $w_{13} = 6$, $w_{23} = 0$, $w_{24} = 5$, $w_{34} = 8$. Select the first node for branching.

Step 2. The maximal vertical distance is 0: update $ub = 34$.

Step 4. Restore the second node at level $p = 4$: $lb = 35 \geqslant ub$. Backtrack to level $p = 3$.

Step 4. Restore the first node at level $p = 3$: $lb = \infty \geqslant ub$. Backtrack to level $p = 2$.

Step 4. Restore the second node at level $p = 2$: $lb = 34 \geqslant ub$. Backtrack to level $p = 1$.

Step 4. Restore the first node at level $p = 1$: $lb = \infty \geqslant ub$. Backtrack to level $p = 0$.

Step 4. $p = 0$; repeat this procedure with $T = 13$.

Computational results

The horizon-varying algorithm has been coded in Visual C++ Version 4.0 under Windows NT 4.0. In order to validate the algorithm we used the well-known problem generator *ProGen* (Kolisch *et al*[8]) to generate 1800 test instances in activity-on-the-node format using the parameter settings in Table 1. *ProGen* uses the well-known coefficient of network complexity (*CNC*)—defined as the number of precedence relations divided by the number of activities (Pascoe[9])—as a measure for characterising the topological structure of a network. For an activity-on-the-node network with a given number of nodes, a higher network 'complexity' results in an increasing number of arcs and therefore in a greater connectedness of the network. Using five settings for the number of activities, six settings for the number of execution modes, two settings for the problem scale (activity duration and activity cost values), and three settings for *CNC*, we obtained 180 problem classes, each consisting of 10 instances.

The 1800 activity-on-the-node instances were transformed into activity-on-the-arc instances using the procedure described in De Reyck and Herroelen.[10] This

Table 1 Parameter settings

Number of activities	10; 20; 30; 40; 50
Number of modes	Fixed at 2; 4; 6 or randomly chosen from the interval [1,3]; [1,7]; [1,11]
Activity durations and costs	Randomly selected from the interval [1,20] or [1,100]
CNC	1.5; 1.8; 2.1

procedure uses the methodology developed by Kamburowski *et al*[11] to generate, from a given activity-on-the-node network, an activity-on-the-arc network with minimum number of nodes and minimum value of the complexity index *CI*. *CI* essentially measures how nearly series-parallel a network is. It is defined as the reduction complexity of a network, that is the number of node reductions sufficient—along with series and parallel reductions—to reduce a two-terminal directed acyclic network to a single edge.[12] The complexity index of a directed acyclic network is computed as the number of nodes in a minimum node cover of its complexity graph. The complexity graph $C(D)$ of a network $D = (N, A)$ is defined as follows: (i, j) is an arc of $C(D)$, if there exists paths $\pi(1, j)$, $\pi(i, n)$, $\pi_1(i, j)$ and $\pi_2(i, j)$ such that $\pi(1, j) \cap \pi_1(i, j) = \{j\}$ and $\pi(i, n) \cap \pi_2(i, j) = \{i\}$. For a detailed discussion of the algorithm for computing *CI*, we refer the reader to De Reyck and Herroelen.[10] These authors observed a positive correlation between the CPU time required to solve a discrete time/cost trade-off problem and *CI*: the higher *CI*, the harder the problem.

Each problem instance was solved using the horizon-varying approach equipped with the first rule (referred to as *P*1) and the second rule (referred to as *P*2) for computing the vertical distance. The results obtained using the exact procedure based on finding the minimum number of node reductions necessary to transform the network to a series-parallel network (referred to as *P*3), developed by Demeulemeester *et al*,[7] are used as a benchmark. *P*3 is one of the best procedures currently available. Moreover, preliminary tests on several of the instances revealed that it outperformed the alternative procedure developed by Demeulemeester *et al*,[7] which is based on minimising the computational effort in enumerating alternative modes through a branch-and-bound tree. Each procedure was allowed to run on a Dell personal computer equipped with a Pentium 133 MHz processor for a maximum CPU time of 200 seconds.

Table 2 represents the average CPU time (in seconds), the average number of nodes generated in the search tree (only where relevant, *P*1 and *P*2), and the average percentage of problems solved to optimality by each of the three

Table 2 Average CPU time, nodes in the search tree and % problems solved to optimality

	Average CPU time (seconds)			*Average number of nodes in the search tree*		*Average % of problems solved optimally*		
	P1	*P2*	*P3*	*P1*	*P2*	*P1*	*P2*	*P3*
All instances	62.09	65.80	104.13	30500	31008	78%	76%	51%
Number of activities								
10	0.34	0.39	0.02	1359	1434	100%	100%	100%
20	19.17	22.95	70.01	18472	20249	99%	98%	73%
30	58.00	65.13	122.99	35813	37404	86%	83%	42%
40	105.40	110.44	151.73	48281	48290	61%	57%	28%
50	127.56	130.10	175.91	48575	47660	41%	40%	14%
Number of modes								
2	3.77	4.72	49.21	2679	2679	100%	100%	79%
4	85.82	90.72	130.79	40390	40116	69%	68%	39%
6	122.20	123.44	151.91	62987	61165	49%	48%	25%
1 to 3	2.63	2.86	27.99	1719	1737	100%	100%	89%
1 to 7	63.30	71.60	118.46	29906	32631	81%	77%	46%
1 to 11	94.85	101.46	146.44	45319	47717	66%	61%	29%
CNC								
1.5	53.55	57.32	77.73	33732	34107	82%	80%	65%
1.8	61.33	65.23	114.18	28402	29218	78%	76%	46%
2.1	71.40	74.86	120.49	29366	29698	73%	71%	43%
Scale								
1 to 20	49.35	52.64	101.95	19594	19218	84%	82%	52%
1 to 100	74.84	78.96	106.31	41406	42660	71%	69%	51%

procedures. The row labelled 'All instances' gives the average results over all 1800 problem instances. Both horizon-varying procedures clearly outperform procedure *P*3 for both performance measures. The best results are obtained by procedure *P*1. The four remaining rows give more detailed results. The results in the row labelled 'Number of activities' show that the three procedures solve all the 10-activity problems very quickly. Procedures *P*1 and *P*2 find the optimal solution for almost all 20-activity and most of the 30-activity problems. Clearly, the higher the number of activities, the higher the average CPU time required and the smaller the percentage of problems solved.

The results in the row labelled 'Number of modes' reveal the negative effect of the number of modes on the efficiency and effectiveness of the three procedures: the higher the number of modes, the more CPU time required and the smaller the percentage of problems solved to optimality. *P*1 and *P*2 optimally solve all 2-mode problems in less than 5 seconds on the average. Moreover, the problems with a fixed number of modes are more difficult to solve by any of the three procedures than the instances where the number of modes is randomly selected from the three corresponding intervals.

The results in the row labelled '*CNC*' indicate the effect of *CNC* on problem complexity: the higher *CNC* (that is the higher the number of arcs (precedence relations) in the original activity-on-the-node network), the more difficult the problem. This effect must be interpreted with some care. The higher *CNC* of the original activity-on-the-node network, the higher the number of nodes and dummy activities in the corresponding activity-on-the-arc network generated by the Kamburowski *et al*[11] procedure, and the higher the resulting *CI*. As shown below, it is mainly the resulting increase in *CI* which makes the problem harder to solve.

The row labelled 'Scale' shows the impact of the problem scale (activity duration and activity cost values). The more extended the scale, the more difficult to solve the instances.

Tables 3, 4 and 5 show the interaction effects between the number of activities and the number of modes for each of the three procedures. The first number in each cell denotes the average required CPU time (in seconds) for

Table 3 Interaction between the number of activities and the number of modes for *P*1

	Number of modes					
Number of activities	*2*	*4*	*6*	*1 to 3*	*1 to 7*	*1 to 11*
10	0.01	0.24	1.13	0.01	0.20	0.46
	100%	100%	100%	100%	100%	100%
20	0.33	10.73	60.67	0.11	7.60	35.60
	100%	100%	95%	100%	100%	100%
30	1.61	71.39	156.44	0.29	47.55	70.74
	100%	93%	40%	100%	98%	85%
40	5.32	153.46	192.75	2.00	104.89	173.96
	100%	48%	10%	100%	73%	37%
50	11.56	193.27	200.00	10.76	156.24	193.50
	100%	5%	0%	100%	35%	7%

Table 4 Interaction between the number of activities and the number of modes for *P*2

	Number of modes					
Number of activities	*2*	*4*	*6*	*1 to 3*	*1 to 7*	*1 to 11*
10	0.01	0.37	1.24	0.01	0.23	0.48
	100%	100%	100%	100%	100%	100%
20	0.35	14.48	67.87	0.12	9.07	45.81
	100%	100%	92%	100%	100%	95%
30	1.85	86.05	155.54	0.29	65.39	81.62
	100%	87%	38%	100%	92%	83%
40	6.55	158.52	192.56	2.06	120.60	182.35
	100%	47%	10%	100%	62%	25%
50	14.83	194.19	200.00	11.81	162.71	197.06
	100%	5%	0%	100%	33%	3%

Table 5 Interaction between the number of activities and the number of modes for *P*3

	Number of modes					
Number of activities	*2*	*4*	*6*	*1 to 3*	*1 to 7*	*1 to 11*
10	0.00	0.01	0.06	0.00	0.01	0.06
	100%	100%	100%	100%	100%	100%
20	0.04	75.21	159.47	0.02	44.74	140.55
	100%	82%	27%	100%	92%	37%
30	6.46	178.76	200.00	0.63	160.49	191.57
	100%	15%	0%	100%	27%	8%
40	101.62	200.00	200.00	21.72	187.02	200.00
	62%	0%	0%	98%	10%	0%
50	137.93	200.00	200.00	117.56	200.00	200.00
	33%	0%	0%	48%	0%	0%

solving the problems in the corresponding class, while the second number denotes the average percentage of problems solved optimally within the CPU time limit.

*P*1 solves all two-mode problems to optimality in promising CPU times. On the average, slightly more than 5 seconds are needed to solve the 40-activity problems and some 12 seconds are needed for the 50-activity problems (Table 3). The effectiveness decreases when the number of activities reaches 40 and the number of modes reaches 4 or more. Most of the 50-activity problems with 4 or more execution modes are, given the CPU time limit, beyond the capabilities of procedure *P*1. *P*2 is a close runner-up (Table 4). *P*3, being very fast on the small 10-activity instances, runs into trouble on the 50-activity, 2-mode problems (slightly more than one third of the problems can be solved to optimality, as shown in Table 5). Using *P*3, problems with 4 modes become very difficult if not impossible to solve within the specified CPU limit, when the number of activities exceeds 30.

It was already mentioned that the coefficient of network complexity, *CNC* seems to have an effect on the complexity of a problem instance: the higher *CNC*, the more complex the problem. As mentioned above, and as already observed by De Reyck and Herroelen,[10] this effect is mainly due to the associated increase in *CI*.

Figure 7 shows the interaction effect of the complexity index *CI* and the number of activities in terms of the average CPU time required to solve the problems in each corresponding class. The results reported here must be interpreted with sufficient care. As mentioned earlier, the problem generator *ProGen* does not allow for the generation of networks satisfying pre-set values of *CI*. We generated the networks in activity-on-the-node format and transformed them into activity-on-the-arc instances with minimum number of nodes and minimal *CI*-value using the methodology described by Kamburowski *et al.*[11] This explains the somewhat fragmentary results. De Reyck and Herroelen[10] encountered similar problems in generating sufficient problem instances over the full range of *CI*-values.

For a given number of activities, the three procedures show an increasing required CPU time as the value of *CI* goes up. This pattern is more pronounced for procedure *P*3. This result, already confirmed by De Reyck and Herroelen,[10] is not a surprise as the *CI*-concept lies at the very heart of the solution logic of procedure *P*3. Moreover, as can be seen from the plots of $|A| = 10$ and $|A| = 20$, *P*3 seems to outperform *P*1 and *P*2 on the instances with small *CI*. On the instances with higher *CI*-values, the more robust procedures *P*1 and *P*2 outperform *P*3.

Conclusions

This paper reports on a new exact solution procedure for problem $1, T|cpm, disc, mu|curve$, namely the discrete time/cost trade-off problem in deterministic activity-on-the-arc networks, where the activities are subject to finish-start precedence relations and where the duration of each activity is a discrete, nonincreasing function of the amount of a single nonrenewable resource committed to it. The objective is to construct the complete and efficient time/cost profile over the set of feasible project durations. The procedure uses a horizon-varying approach based on the iterative optimal solution of problem 1, $T|cpm, \delta_n, disc, mu|av$, that is the problem of minimising the sum of the resource use over all activities subject to the activity precedence constraints and a project deadline. This optimal solution is derived using a branch-and-bound procedure which computes lower bounds by making convex piecewise linear underestimations of the discrete time/cost trade-off curves of the activities to be used as input for an adapted version of the Fulkerson labelling algorithm for the linear time/cost trade-off problem. Branching involves the selection of an activity in order to partition its set of execution modes into two subsets which are used to derive improved convex piecewise linear underestimations. The procedure has been programmed in

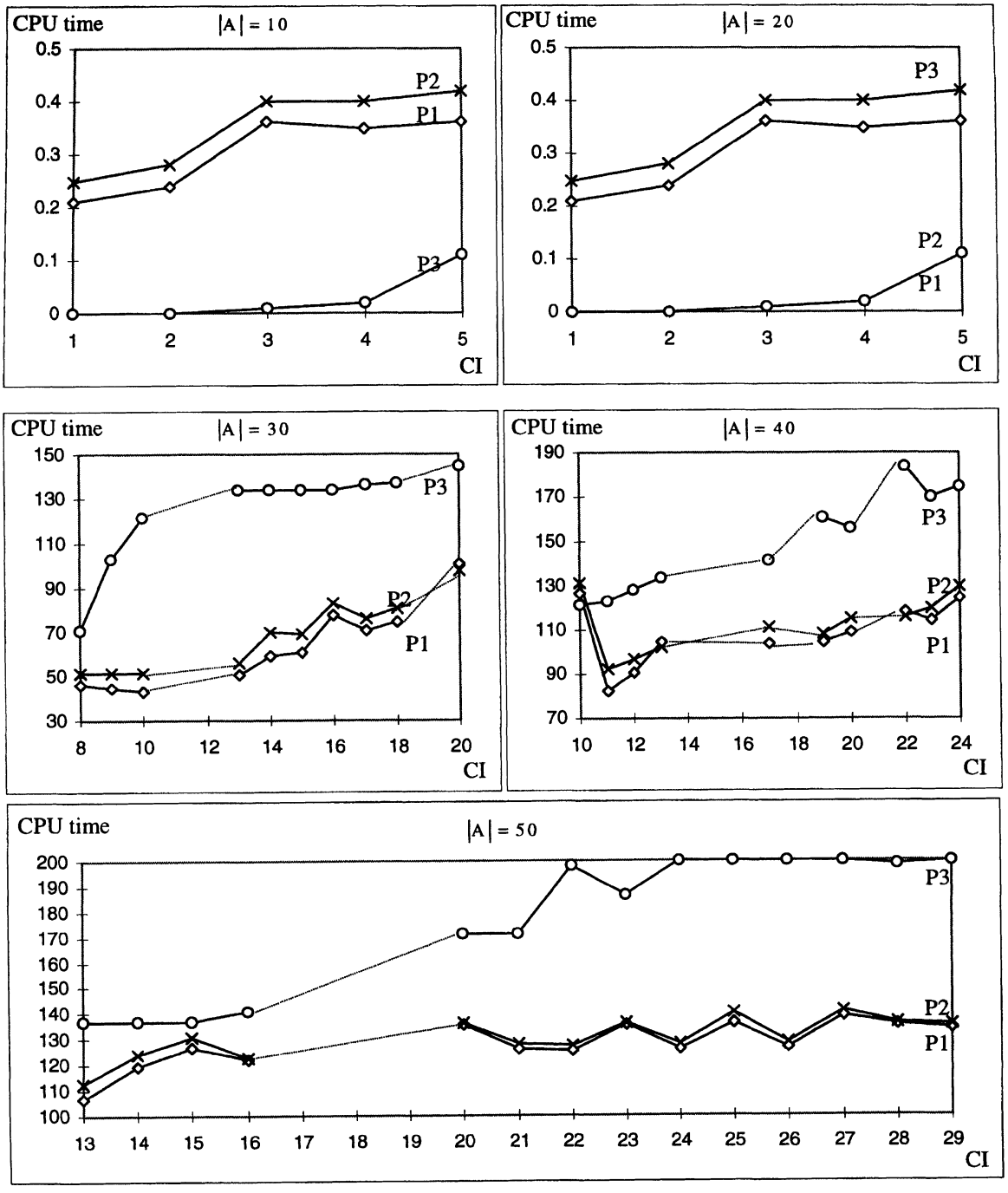

Figure 7 Interaction between *CI* and the number of activities.

Visual C++ under Windows NT for use on personal computers and has been validated using a factorial experiment on a large set of problem instances.

The results obtained are encouraging, both horizon-varying procedures clearly outperform procedure *P*3 developed earlier by Demeulemeester *et al*[7] in terms of the average required CPU time and the average percentage of problems solved optimally. The best results are obtained by procedure *P*1. The three procedures optimally solve all the 10-activity problems at very small average CPU times. Procedures *P*1 and *P*2 find the optimal solution for almost all 20-activity and most of the 30-activity problems in acceptable time. Clearly, the number of activities and the number of modes have a negative effect on both the effectiveness and the efficiency of the procedures. On the average, problems with more than fifty activities and six or more modes resist an optimal solution in fifty or more percent of the tested cases.

Although the rather fragmentary results on the impact of the complexity index must be interpreted with sufficient care, it was found that, for a given number of activities, the three procedures show an increasing overall pattern of required CPU times as the value of *CI* goes up. This pattern seems to be more pronounced for procedure *P*3. This result, already confirmed by Demeulemeester *et al*,[7] is not a surprise as the *CI*-concept lies at the very heart of the solution logic of procedure *P*3.

Acknowledgements—This research was supported by the National Science Foundation under Research Contract FWO No. G.0131.96.

References

1 Moder JJ, Phillips CR and Davis EW (1983). *Project Management with CPM, PERT and Precedence Diagramming*, 3rd ed. Van Nostrand Reinhold: New York.

2 De P, Dunne EJ, Ghosh JB and Wells CE (1995). The discrete time/cost trade-off problem revisited. *Eur J Opl Res* **81**: 225–238.

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

→ Fulkerson DR (1961). A network flow computation for project cost curves. *Mgmt Sci* **7**: 167–179.

5 Ahuja RK, Magnanti TL and Orlin JB (1989). Network flows. In: Nemhauser G *et al* (eds). *Optimisation, Handbooks in Operations Research and Management Science*, Vol. 1. North-Holland, Chapter 4, pp 211–369.

→ De P, Dunne EJ, Ghosh JB and Wells CE (1997). Complexity of the discrete time/cost trade-off problem for project networks. *Ops Res* **45**: 302–306.

7 Demeulemeester EL, Herroelen WS and Elmaghraby SE (1996). Optimal procedures for the discrete time/cost trade-off problem in project networks. *Eur J Opl Res* **88**: 50–68.

→ Kolisch R, Sprecher A and Drexl A (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Mgmt Sci* **41**: 1693–1703.

9 Pascoe TL (1966). Allocation of resources—CPM. *Rev fr Rech opér* **38**: 31–38.

10 De Reyck B and Herroelen W (1966). On the use of the complexity index as a measure of complexity in activity networks. *Eur J Opl Res* **91**: 347–366.

11 Kamburowski J, Michael DJ and Stallmann MFM (1992). Optimal construction of project activity networks. *Proc 1992 Annual Meeting of the Decision Sciences Institute*, Decision Sciences Institute, USA pp 1424–1426.

12 Bein WW, Kamburowski J and Stallmann MFM (1992). Optimal reduction of two-terminal directed acyclic grarphs. *SIAM J Computing* **21**: 1112–1129.

Received December 1997;
accepted July 1998 after one revision