New computational results for the discrete time/ cost trade-off problem with time-switch constraints

Vlerick Leuven Gent Management School
Associated with Ghent University and with the Katholieke Universiteit Leuven

Vlerick Working Papers 2002/18

NEW COMPUTATIONAL RESULTS FOR THE DISCRETE TIME/COST TRADE-OFF PROBLEM WITH TIME-SWITCH CONSTRAINTS

MARIO VANHOUCKE

e-mail: mario.vanhoucke@vlerick.be

D/2002/6482/19

NEW COMPUTATIONAL RESULTS FOR THE DISCRETE TIME/COST TRADE-OFF PROBLEM WITH TIME-SWITCH CONSTRAINTS

MARIO VANHOUCKE

e-mail: mario.vanhoucke@vlerick.be

Operations and Technology Management Center

Vlerick Leuven Gent Management School

Bellevue 6, B-9050 Gent (Belgium)

e-mail: mario.vanhoucke@rug.ac.be

Department of Management Information, Operations Management and Technology Policy

Faculty of Economics and Business Administration, Ghent University

Hoveniersberg 24, B-9000 Gent (Belgium)

ABSTRACT

Recently, time-switch constraints have been introduced in literature by Yang and Chen (2000). Basically, these constraints impose a specified starting time on the project activities and force them to be inactive during specified time periods. This type of constraints have been incorporated into the well-known discrete time/cost trade-off problem in order to cope with day, night and weekend shifts.

In this paper, we propose a new branch-and-bound algorithm which outperforms the previous one by Vanhoucke et al. (2002a). The procedure makes use of a lower bound calculation for the discrete time/cost trade-off problem (without time-switch constraints). The procedure has been coded in Visual C++, version 6.0 under Windows 2000 and has been validated on a randomly generated problem set.

Keywords: *Project Management; CPM; Time/cost trade-off problem; Time-switch constraints.*

1. INTRODUCTION

Time/cost trade-offs in project networks have been the subject of intensive research since the development of the critical path method (CPM) in the late 50s. Time/cost behaviour in a project activity basically describes the trade-off between the duration of the activity and its amount of nonrenewable resources (e.g. money) committed to it. It is generally accepted that the trade-off follows a nonincreasing pattern, i.e. expediting an activity is possible by allocating more resources (i.e. at a larger cost) to it. However, due to its complexity (the problem is known to be NP hard (see De et al. (1997)), the problem has been studied under a number of different assumptions.

The early time/cost trade-off models assumed the direct activity cost functions to be *linear* non-increasing functions. The objective was to determine the activity durations and to schedule the activities in order to minimize the project costs, i.e. the sum of the direct activity and the time-dependent indirect project costs, within a specified project deadline. Therefore, the activity costs are a function of the activity durations, which are bounded from below (crash duration) and from above (normal duration). Solution procedures for the linear case are proposed by Kelley and Walker (1959), Fulkerson (1961), Kelley (1961), Ford and Fulkerson (1962), Siemens (1971), Goyal (1975) and Elmaghraby and Salem (1981). Several other forms of activity cost functions have been studied, such as *concave* (Falk and Horowitz (1972)), *convex* (Lamberson and Hocking (1970), Kapur (1973), Siemens and Gooding (1975), Elmaghraby and Salem (1980a,b)) or even *general* continuous activity cost functions (Moder et al. (1983)).

As research efforts progressed and practical needs arose, researchers began to focus on the development of procedures for solving the *discrete* version of the problem. This discrete time/cost trade-off problem occurs when the duration of project activities is a discrete, nonincreasing function of the amount of a single nonrenewable resource committed to them. It involves the selection of a set of execution modes (the time-cost tuples for each activity) in order to achieve a certain objective. In the literature, the problem objective has been divided into three parts. The so-called *deadline* problem (problem $1,T|cpm,\delta_n,disc,mu|av$ following the classification scheme of Herroelen et al. (1999)) aims at minimizing the total cost of the project while meeting a given deadline while the *budget* problem (problem $1,T|cpm,disc,mu|C_{max}$) involves minimizing the project duration without exceeding a given budget. A third objective is to construct the complete and efficient time/cost profile over the set of feasible project durations (problem $1,T|cpm,disc,mu|curve$). Research papers has been written by Crowston and Thompson (1967), Crowston (1970), Robinson (1975) Billstein and Radermacher (1977), Wiest and Levy (1977), Hindelang and Muth (1979), Patterson and Harvey (1979), Bianco and Speranza (1990), Vercellis (1990), Elmaghraby and Kamburowski (1992), De et al. (1995, 1997), Demeulemeester et al. (1996,

1998), Skutella (1998) and Akkan et al. (2000a,b)). In an attempt to tighten the bridge with the current practice, Vanhoucke et al. (2002a) have extended this discrete problem type with so-called time-switch constraints, as introduced by Yang and Chen (2000).

Although the first research endeavors stem from more than 50 years ago, the problem still is very actual. Indeed, next to the famous project examples (e.g. the tunnel between France and the United Kingdom), production managers also focus on the development of unique products, such as the development of a new automation system, the installation of a new software program, and many others. Moreover, due to the increasing competition, many firms are obliged to accomplish tasks that do not fit neatly into business-as-usual. Project management fits very well into this new philosophy, due to its unique characteristics and its known and limited duration. Finally, in our current environment of time-based competition, expediting activities is a matter of course, which defends the use of time/cost trade-offs on the activity level.

In this paper, we focus on the discrete time/cost trade-off problem with time-switch constraints for which the literature is, to the best of our knowledge, restricted to the paper by Vanhoucke et al. (2002a). These constraints impose specific starting times on the project activities and force them to be active and inactive during specific time periods. Consequently, these types of constraints serve well for incorporating day and night shifts. In the following section we briefly review the features of the problem and the branch-and-bound approach of Vanhoucke et al. (2002a). In section 3 we describe our new exact procedure. We also illustrate the procedures using a real-life problem example. A section is reserved on detailed computational results. The last section draws overall conclusions.

2. DESCRIPTION OF THE PROBLEM

In the sequel of this paper, we assume that a project is represented by an activity-on-the-arc network $G=(N,A)$ where the set of nodes, N, represents network events and the set of arcs, A, represents the activities of the project. The nodes of the network are numbered from the single start node 1 to the single end node n. The duration x_i of an activity $i \in A$ is a discrete, nonincreasing function of the amount of a single nonrenewable resource (money, y_i) allocated to it. The tuple (x_i, y_i) is referred to as a *mode*. The deadline version of the *discrete time/cost trade-off problem* without time-switch constraints (problem $1,T|cpm,\delta_n,disc,mu|av$) involves the scheduling of project activities in order to minimize the total cost of the project.

Vanhoucke et al. (2002a) have extended this basic problem type with so-called time-switch constraints. These constraints force activities to start in a specific time interval and to be down in some specified rest intervals. In this paper, they have restricted their analysis to three different

work/rest patterns. For an overview, we refer to the paper by Yang and Chen (2000), in which the concept 'time-switch constraint' has been introduced for the very first time.

The specific time-switch constraints addressed in this paper can be classified into three main categories, corresponding to three different shifts. An activity following the so-called *day*-pattern can only be executed during day time, from Monday till Friday. This pattern may be imposed when many persons are involved in executing the activity. Activities that follow a *d&n*-pattern can be executed during the day or night, from Monday till Friday. This pattern may be followed in situations where activities require only one person who has to control the execution of the activity once and a while. A *dnw*-pattern consists of activities that can be in execution every day or night and also during the weekend. This may be the case for activities which do not require human intermission.

Furthermore, we assume that an activity duration is expressed in work periods of 12 hours. Moreover, one time unit equals 24 hours and is equal to two works periods. Figure 1 illustrates the gannt chart for three activities with a duration of 4 work periods under the three different time-switch constraints. Black areas represent work periods while grey areas symbolize rest periods. Finally, the lead time for an activity equals the finishing time f_i – starting time s_i. For more details, we refer the reader to Vanhoucke et al. (2002a). In the sequel of this paper we assume that each activity starts as early as possible within the precedence constraints and, consequently, no further ready times are imposed on the activities. However, an overall specified starting time of the whole project is imposed, varying from Monday till Sunday.

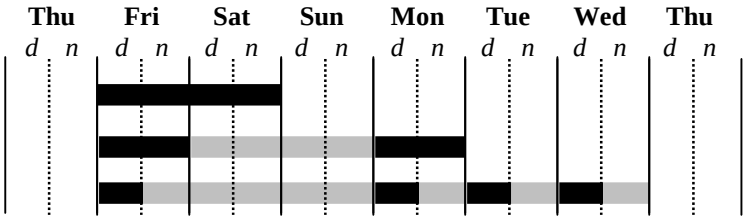

Figure 1. *A Gannt chart for activities with a duration of 4 work periods*

According to the classification scheme of Herroelen et al. (1999), the discrete time/cost trade-off problem with time-switch constraints (*DTCTPTSC*) can be referred to as problem $1,T|tsc,cpm,\delta_n,disc,mu|av$. Since time-switch constraints (*tsc*) can be considered as a variant of preemption (work periods alternated by rest periods), we have placed the abreviation *tsc* in the second field (the so-called β-field).

3. SOLUTION PROCEDURES FOR THE PROBLEM

In section 3.1 we briefly discuss the logic of the branch-and-bound algorithm of Vanhoucke et al. (2002a). In sections 3.2 and 3.3, we present a completely new branch-and-bound procedure that clearly overcomes the drawbacks of the previous procedure, as shown in our computational results section. In both approaches we make use of the branch-and-bound procedure of Demeulemeester et al. (1998) for calculating the lower bounds.

3.1. The exact branch-and-bound procedure of Vanhoucke et al. (2002a)

In the sequel of this paper, we refer to the paper by Vanhoucke et al. (2002a) as the *VDH* procedure. This branch-and-bound procedure is based on the following logic: at the root node of the branch-and-bound tree, all activities are scheduled in their normal mode (i.e. the mode corresponding to the highest duration and the lowest cost). If this schedule has a duration which is lower than or equal to the project deadline, the algorithm reports the optimal cost of the project. If this is not the case, the algorithm starts to branch.

The branching strategy identifies critical activities which have to be crashed in order to decrease the project duration. To that purpose, the algorithm randomly selects a critical path and generates a new node on the next level of the branch-and-bound tree for each activity of this critical path. Activities of the critical path are crashed at each new node of the branch-and-bound tree. At each node, lower bounds are calculated on the total project cost by the procedure of Demeulemeester et al. (1998). If this lower bound solution corresponds to a feasible schedule (i.e. the project duration with time-switch constraints is smaller than or equal to the deadline), the algorithm updates the upper bound *ub* of the project. Nodes are pruned due to lower bounds that are larger than the current feasible solution or due to the application of an adapted version of the subset dominance rule (De Reyck and Herroelen (1998)). The algorithm backtracks when there are no nodes left unexplored at a particular level and stops when it backtracks to the initial level of the branch-and-bound tree.

The computation of the lower bounds at each node is based on the procedure of Demeulemeester et al. (1998) for the discrete time/cost trade-off problem without time-switch constraints (further abbreviated as the ***DTCTP***). To that purpose, the activity durations of all the modes must be modified by adding the *minimal rest time* of each activity. This minimal rest time corresponds to the rest time of each activity under the assumption that this activity would start on Monday.

As shown in the *VDH* paper, the new duration x'_i for each mode of each non-dummy activity *i* can be calculated by adding the minimal rest time according to the following formulas:

day-pattern : $x'_i = 2\,x_i + 4\lfloor(x_i - 1)/5\rfloor$ [1]

d&n-pattern : $x'_i = x_i + 4\lfloor(x_i - 1)/10\rfloor$ [2]

dnw-pattern : $x'_i = x_i$ [3]

which modifies the new duration x'_i of each activity to its lead time when it starts on Monday. Note that we assume that no activity can be started during a night period.

3.2. Our new branch-and-bound procedure

Our new branch-and-bound algorithm uses an identical approach to calculate the lower bound at each node of the tree. Consequently, the lower bound algorithm ignores the time-switch constraints and reports an optimal mode for each activity with a corresponding *lb*-cost and a project duration. If the corresponding time-switch schedule (i.e. scheduling all activities with the *lb*-modes without ignoring the time-switch constraints) results in a total project duration larger than the project deadline, the algorithm needs to branch. This branching strategy differs completely from the *VDH* procedure.

3.2.1. The branching strategy

The algorithm searches for a branching activity *ba* for which the lead time is larger than its duration x'_i under the lower bound calculations. In figure 2 we illustrate the potential difference between the lower bound duration x'_i and the corresponding time-switch lead time for an activity following a day-pattern with a duration of 4 work periods. The minimal duration of this activity is – according to equation [1] – $x'_i = 2 * 4 + 4\lfloor(4 - 1)/5\rfloor = 8$. However, figure 2 reveals that this duration is only valid for schedules (1) and (2) when the activity starts on Monday or Tuesday. The lead time of this activity amounts to 12 work periods for schedules (3) – (6) and 10 for schedule (7).

Consequently, depending on the start time of the activity, it is possible that the lead time is larger than the activity duration x'_i used in the lower bound calculations.

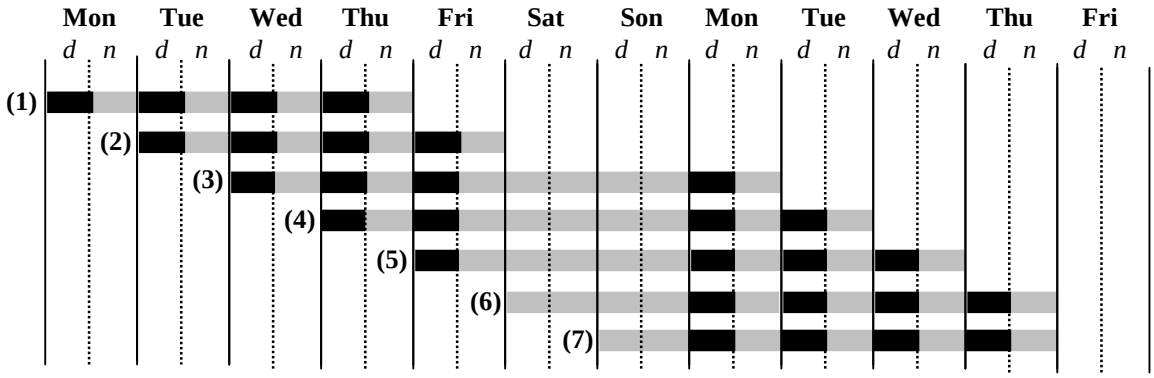

Figure 2. *A day activity with duration of 4 work periods*

Our new branching strategy relies completely on this observation and selects an arc out of the network as the current branching activity *ba* (see figure 3). This branching activity must have a lead time which is larger than the reported activity duration, given its starting time reported by the *DTCTP* algorithm (lower bound calculation). The algorithm then divides the possible starting times of the branching activity into three distinctive regions. To that purpose, the algorithm computes two possible starting times t_1 and t_2. Suppose e.g. that the *DTCTP* algorithm reports that an activity *ba* starts at the 18th day (work period 36, a Thursday) of a project starting on Monday, January 1st. Consequently, this activity behaves like schedule (4) of figure 2 with a lead time of 12 work periods which is larger than the duration of 8 work periods. This lead time is valid between work period t_1 = 34 (Wednesday) and t_2 = 40 (Saturday). Between these two starting times, the activity duration is set to x'_i = 12, outside this range, we still use the duration of x'_i = 8.

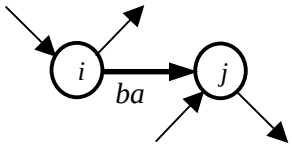

Figure 3. *The branching activity ba in a project network*

In doing so, we create three new child nodes at the next level of the branch-and-bound tree and divide the possible starting times of the branching activity into three distinct regions. The characteristics of the three nodes can be described as follows:

Node 1. Event *i* must start earlier than t_1

At the first child node of the branch-and-bound tree, the algorithm forces the branching activity $ba = (i, j)$ to start earlier than the reported period t_1. To that purpose, we simply add an extra arc from event *i* (i.e. the start node of *ba*) to the dummy end node of the project, as shown in figure 4. This extra arc has only one mode with duration = $\delta_n - t_1 + 2$ and a zero cost. Since we force to start *ba* outside the range $[t_1, t_2]$, the algorithm calculates a lower bound with the *DTCTP* procedure without changing the duration of *ba*, i.e. $x'_{ba} = 8$.

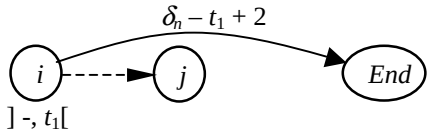

Figure 4. *Start ba earlier than time period* t_1

Node 2. Event *i* must start later than t_2

Similar to node 1 of the branch-and-bound tree, the algorithm now forces event *i* to start later than the reported period t_2. As illustrated in figure 5, this can be done by adding an extra arc between the dummy start node of the project and event *i*. This single mode activity has a duration of $t_2 + 2$ with a zero cost. The duration of *ba* during the execution of the lower bound calculation remains unchanged, i.e. $x'_{ba} = 8$.

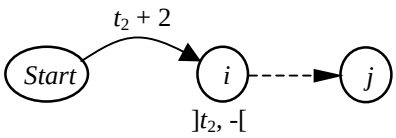

Figure 5. *Start ba later than time period* t_2

Node 3. Event *i* must start between t_1 and t_2

The third child node is somewhat different than the two previous nodes of the tree. In this node the algorithm forces the branching activity to start in the time interval [t_1, t_2] by adding two extra arcs as presented in figure 6. In fact, in the parent node of our branch-and-bound tree, the lower bound procedure reported a starting time for *ba* between t_1 and t_2. The duration x'_{ba}, however, was too small since the lead time turned out to be larger. To that purpose, we alter the duration at this node from $x'_{ba} = 8$ to $x'_{ba} = 12$ in our example. The two extra arcs (start, *i*) and (*i*, end) have a zero cost and a duration of t_1 and $\delta_n - t_2$, respectively.

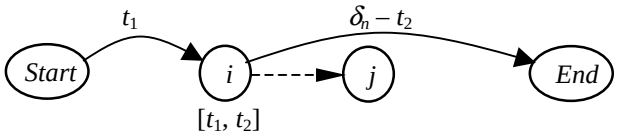

Figure 6. *Start ba between time periods t_1 and t_2*

3.2.2 The branch-and-bound tree

As a result of the previous subsection, each parent node in our branch-and-bound tree has three child nodes which differ from the parent node by the incorporation of one or two extra arcs (see figure 7). B&B nodes can be frequently fathomed due to the incorporation of an arc in the project network leading to infeasible solutions. Moreover, B&B nodes with a lower bound exceeding the current upper bound of total project cost (initially set to ∞) can also be fathomed. Otherwise, the algorithm continues with the node with the smallest lower bound. If the lower bound solution reported by the *DTCTP* algorithm corresponds with a feasible solution, the upper bound is updated. A feasible solution corresponds with a time-switch-schedule for which all activities are scheduled in their mode as reported by the *DTCTP* procedure and for which the project duration is smaller than or equal to the negotiated project deadline. If there are no nodes left unexplored at a particular level, the algorithm backtracks to the previous level and continues with the remaining unexplored node with the smallest lower bound. The algorithm stops when it backtracks to the initial level of the branch-and-bound tree.

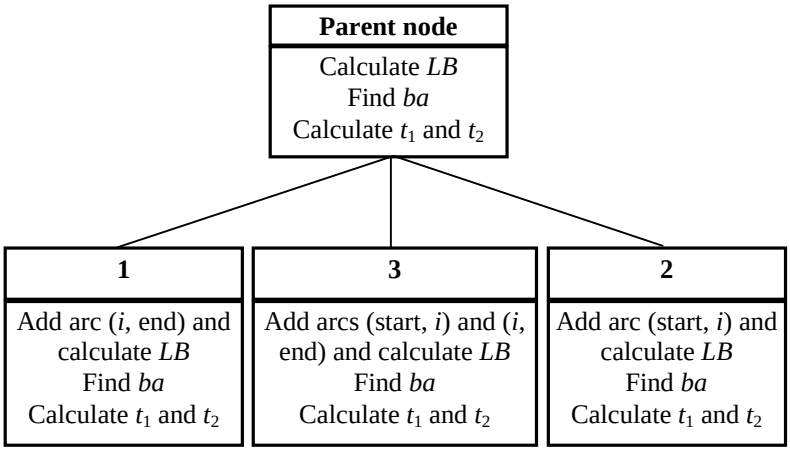

Figure 7. *The three different child nodes used in the branch-and-bound tree*

3.2.3. Reducing the number of nodes of the branch-and-bound tree

As previously described, the algorithm generates three child nodes at each level of the branch-and-bound tree in which the allowable realization time of certain project event nodes has been restricted. To that purpose, the algorithm calculates two time periods t_1 and t_2 and restricts the range to]-, t_1[for the first child node,]t_2, -[for the second child node and [t_1, t_2] for the third child node. In this section, we modify the allowable ranges]-, t_1[, [t_1, t_2] and]t_2, -[to]-, t_1"], [t_1', t_2'] and [t_2", -[such that $t_1" < t_1 \leq t_1' \leq t_2' \leq t_2 < t_2"$ (see figure 8). This modification leads to a decrease in the number of nodes in the search tree. Moreover, it allows the algorithm to prune certain nodes of the branch-and-bound tree.

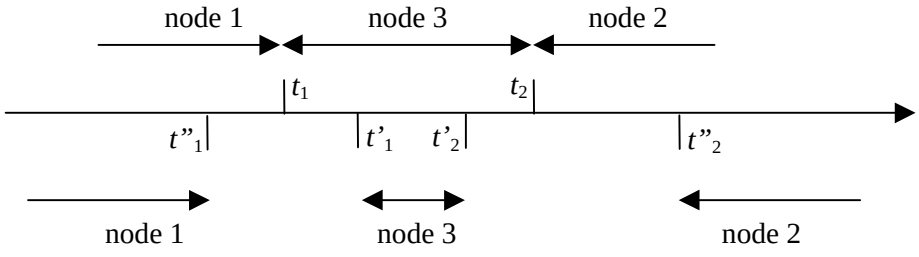

Figure 8. *Modification of the allowable ranges that determine nodes* 1, 2 *and* 3 *in the tree*

In order to modify the allowable ranges of the starting event *i* of a branching activity at each level, we need to calculate two sets as a basis for our modifications. Therefore, FR_i denotes the set of possible realization times of event *i* by means of forward calculations (starting from time zero) and BR_i represents the set of possible realization times of event *i* by means of backward calculations (starting from the project deadline). These two sets are simply the result of forward and backward calculations of each event node in the network taking the time-switch constraints into account. Consequently, the set FR_j consists of all the possible finishing times of all arcs (*i*, *j*), i.e. the time-switch finishing time for each possible duration and for each possible start time (as given by FR_i). Notice that these two sets might change along the exploration of the branch-and-bound tree due to the incorporation of extra arcs.

In the remaining of this section, we discuss the modifications of the allowable ranges for the three nodes at each level.

Node 3. Transformation of $[t_1, t_2]$ to $[t_1', t_2']$

The two sets FR_i and BR_i can be used for the transformation of the interval $[t_1, t_2]$ to the interval $[t_1', t_2']$ (with $t_1 \leq t_1' \leq t_2' \leq t_2$) as follows: t_1' = minimum$\{z_j | z_j \geq t_1, z_j \in FR_i$ or $z_j \in BR_i\}$ and t_2' = maximum$\{z_j | z_j \leq t_2, z_j \in FR_i$ or $z_j \in BR_i\}$. Apart from a possible reduction in the total number of nodes in the search tree, this transformation allows us to present a new dominance rule for node 3 at each level.

Dominance rule: If the new range $[t_1', t_2']$ has no time periods in common with FR_i or BR_i (i.e. $[t_1', t_2'] \cap FR_i = \varnothing$ or $[t_1', t_2'] \cap BR_i = \varnothing$) then the exploration of the branch-and-bound node with range $[t_1', t_2']$ can be fathomed.

The logic of this dominance rule is that all possible schedules found by node 3 can also be found by exploring either nodes 1 or 2. Indeed, when we force an activity to start later than t_2' during our branch-and-bound search, this does not imply that this activity cannot start earlier than t_2' in the final optimal solution. Our branch-and-bound procedure is developed to select that mode for each activity that leads to an optimal solution. In constructing the final schedule, given the optimal mode combinations, we use a *semi-active timetable* in which the activity under study can possibly start earlier than t_2'. Consequently, the dominance rule boils down to the fact that set of possible mode combinations of the problem by exploring node 3 is a subset of the set of possible solutions by exploring either node 1 or node 2. Therefore, it can never be advantageous to explore

both nodes 1 and 3 or nodes 2 and 3 is this case. This basic observation will be illustrated in detail by means of the example of section 4 and the appendix.

In the following we discuss the transformations of nodes 1 and 2. Note that the transformation of the allowable ranges at nodes 1 and 2 depend on the application of the dominance rule.

Node 1. Transformation of]-, t_1[to]-, t_1"]

The set FR_i can be used for the transformation of the interval]-, t_1[to the interval]-, t_1"] (with $t_1" < t_1$) as follows: $t_1"$ = maximum$\{z_j | z_j < t_1$ and $z_j \in FR_i\}$. In doing so, we restrict the allowable range which leads to a possible reduction in the total number of nodes in the search tree.

Node 2. Transformation of]t_2, -[to [t_2", -[

Both sets FR_i and BR_i can be used for the transformation of the interval]t_2, -[to the interval [t_2", -[(with $t_2" > t_2$). The value of $t_2"$ depends on the application of the dominance rule described earlier. A global overview is given in table 1.

Table 1. *Possible transformations of*]t_2, -[*to* [t_2", -[*depending on the behaviour of range* [t_1', t_2']

Node 3	**Node 2**
(*a*) [t_1', t_2'] $\cap$ FR_i = $\varnothing$ and [t_1', t_2'] $\cap$ BR_i = $\varnothing$	$t_2" = \max\{\min(z_j \mid z_j \in FR_i$ and $z_j > t_2), \min(z_k \mid z_k \in BR_i$ and $z_k > t_2)\}$
(*b*) [t_1', t_2'] $\cap$ $FR_i \neq \varnothing$ and [t_1', t_2'] $\cap$ BR_i = $\varnothing$	$t_2" = \min\{z_j \mid z_j \in BR_i$ and $z_j > t_2\}$
(*c*) [t_1', t_2'] $\cap$ FR_i = $\varnothing$ and [t_1', t_2'] $\cap$ $BR_i \neq \varnothing$	$t_2" = \max\{\min(z_j \mid z_j \in FR_i$ and $z_j > t_2), \min(z_k \mid z_k \in BR_i$ and $z_k > t_2)\}$
(*d*) [t_1', t_2'] $\cap$ $FR_i \neq \varnothing$ and [t_1', t_2'] $\cap$ $BR_i \neq \varnothing$	$t_2" = \max\{\min(z_j \mid z_j \in FR_i$ and $z_j > t_2), \min(z_k \mid z_k \in BR_i$ and $z_k > t_2)\}$

Remark that the dominance rule for node 3 can only be applied in cases (*a*), (*b*) and (*c*). The transformation of t_2 to $t_2"$ with $t_2" = \max\{\min(z_j | z_j \in FR_i$ and $z_j > t_2), \min(z_k | z_k \in BR_i$ and $z_k > t_2)\}$ is only valid for cases (*a*), (*c*) and (*d*). In case (*b*) we have to calculate the value of $t_2"$ in a different

way since otherwise the optimal solution can be excluded of our search. In the appendix we illustrate this with a small example.

4. A NUMERICAL EXAMPLE FOR THE DTCTPTSC

In this section we illustrate our new branch-and-bound algorithm for the discrete time/cost trade-off problem with time-switch constraints by means of the example AoA network of Vanhoucke et al. (2002a). In doing so, we are able to compare the branch-and-bound tree of both approaches. This project is based on a real-life situation in which certain installations have to be cleaned and contains 20 non-dummy activities (see figure 9.

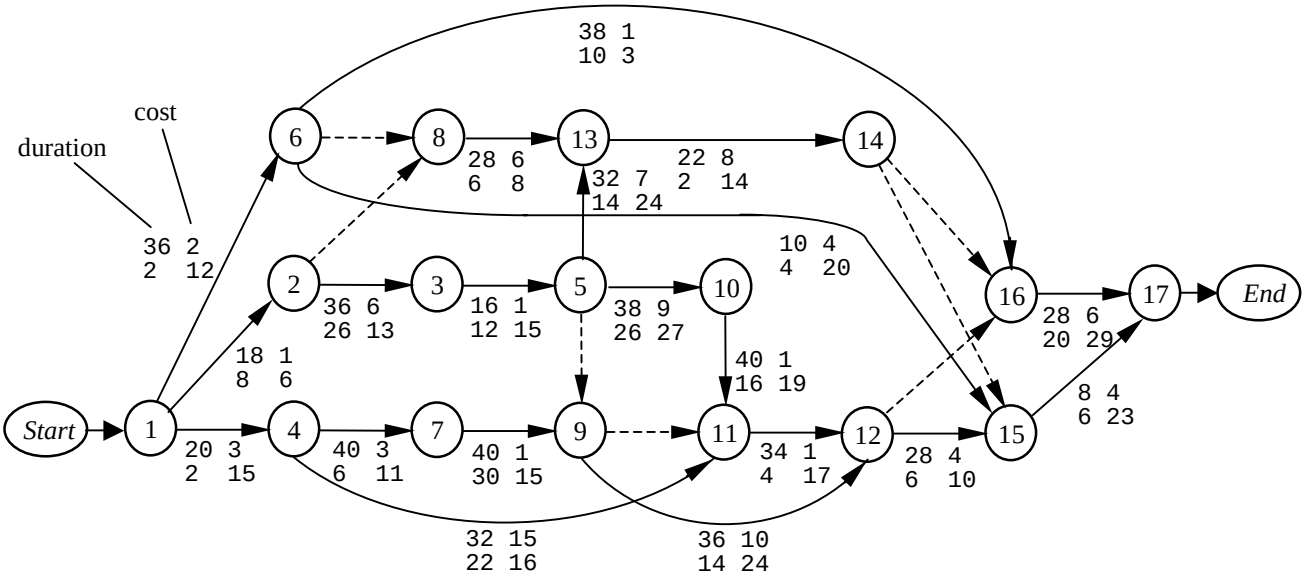

Figure 9. *The example project network of Vanhoucke et al. (2002a)*

Each arc has been assigned a unique activity character. Dashed arcs are used to denote the dummy arcs. Since this network is only for illustrative purposes, we only use two activity modes for each activity, which are displayed near the arcs. Arc (1,6), for example, has a normal duration of 36 work periods and a corresponding cost of 2 or a crash duration of 2 work periods with a corresponding cost of 12. Each activity follows one of the three patterns as described in table 2. We assume that the project has to start on a Monday and the project deadline δ_n amounts to 143 time units (or 286 work periods), which is the average of the earliest possible project finishing time (88 time units) and the latest allowable project finishing time (198 time units).

Table 2. *Details of the example project network of figure* 9

Act	Arc	Pattern	Act	Arc	Pattern	Act	Arc	Pattern
A	(1,2)	*dnw*	**J**	(5,10)	*dnw*	**S**	(10,11)	*d&n*
B	(1,4)	*dnw*	**K**	(5,13)	*day*	**T**	(11,12)	*d&n*
C	(1,6)	*d&n*	**L**	(6,8)	*dummy*	**U**	(12,15)	*day*
D	(2,3)	*d&n*	**M**	(6,15)	*d&n*	**V**	(12,16)	*dummy*
E	(2,8)	*dummy*	**N**	(6,16)	*dnw*	**W**	(13,14)	*day*
F	(3,5)	*dnw*	**O**	(7,9)	*day*	**X**	(14,15)	*dummy*
G	(4,7)	*day*	**P**	(8,13)	*day*	**Y**	(14,16)	*dummy*
H	(4,11)	*dnw*	**Q**	(9,11)	*dummy*	**Z**	(15,17)	*day*
I	(5,9)	*dummy*	**R**	(9,12)	*d&n*	**Γ**	(16,17)	*day*

The branch-and-bound solution

The branch-and-bound tree for the example is displayed in figure 11. The number in bold on the first line denotes the number of each node in the branch-and-bound tree. The numbers on the second line, separated by a slash, denote the lower bound on the total project cost and the project duration with time-switch constraints in time units. A total project duration which is lower than or equal to the deadline corresponds to a feasible node in the tree. The numbers below denote the allowable range of project event *i* (starting event of the branching activity) due to the incorporation of an extra arc at this node of the branch-and-bound tree.

As explained in previous sections, the variable x_i denotes the duration of activity *i* in its current mode. Moreover the variable x'_i is used to denote the adapted duration of activity *i* which is used as an input duration for the lower bound calculation by the procedure of Demeulemeester et al. (1998) for the *DTCTP*. As an example, the normal durations of each activity at the initial node of the tree are, due to the calculations of Eqs. [1]-[3], $x'_A = 18$, $x'_B = 20$, $x'_C = 48$, $x'_D = 48$, $x'_E = 0$, $x'_F = 16$, $x'_G = 108$, $x'_H = 32$, $x'_I = 0$, $x'_J = 38$, $x'_K = 88$, $x'_L = 0$, $x'_M = 10$, $x'_N = 38$, $x'_O = 108$, $x'_P = 76$, $x'_Q = 0$, $x'_R = 48$, $x'_S = 52$, $x'_T = 46$, $x'_U = 76$, $x'_V = 0$, $x'_W = 60$, $x'_X = 0$, $x'_Y = 0$, $x'_Z = 20$ and $x'_\Gamma = 76$.

This set of durations is used as an input for the lower bound calculations at the initial node of the tree. The corresponding lower bound on the total project cost amounts to 118 with a project duration of 198 time units (this is the project duration for the *DTCTPTSC* and *not* the project duration that results from the lower bound calculation). Since this project duration exceeds the project deadline (δ_n = 143 time units), the algorithm identifies a branching activity *ba* for which its lead time is larger than its duration x'_{ba}. Activity C (arc (2,3)) is scheduled to start at work period 8 and consequently, its lead time (52 work periods) is larger than its duration x'_B = 48 for that mode. This lead time of 52 holds within the range $[t_1, t_2]$ = [6, 10].

The set FR_2 can simply be calculated by hand using forward calculations and equals {8, 18}. The set BR_2 = {8, 14, 16, …, 132, 134} (this is somewhat more difficult to see at a single glance). Consequently, the restricted range $[t_1', t_2']$ equals [8, 8] using the formulas as described in section 3.2.3. Since $[t_1', t_2'] \cap FR_2 \cap BR_2 \neq \varnothing$, the algorithm cannot fathom node 3 of the branch-and-bound tree. The range of node 4 equals [18, -[according to section (*d*) of table 1. Node 2 can be fathomed since it has no feasible solution. The algorithm chooses the node with the lowest lower bound as the first branching node, i.e. node 3, to generate three new nodes at the next level of the branch-and-bound tree. The branching activity of this node is *W* (arc (13, 14)) since this activity has been scheduled on time period 150 (Saturday) with a lead time of 64, larger than its duration of 60 work periods. Since this branching activity can be used to illustrate the use of the dominance rule, we illustrate the schedule of this arc in figure 10.

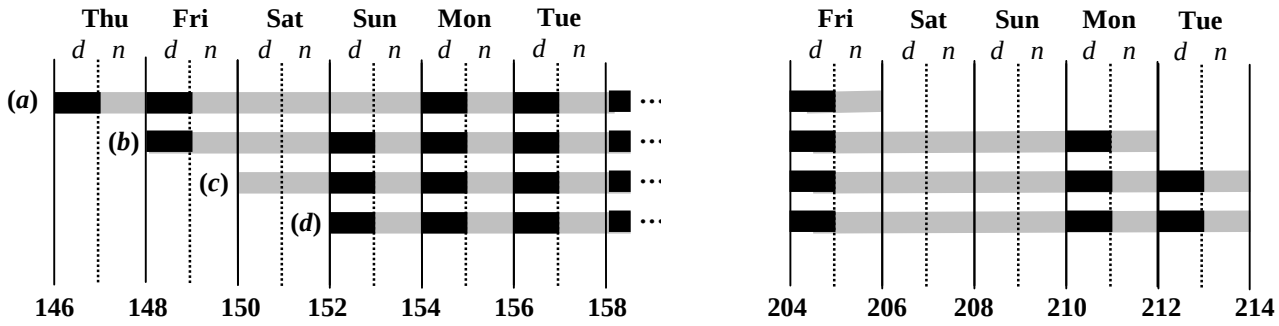

Figure 10. *Possible schedules of a branching activity for the example network*

The lead times of (*a*), (*b*), (*c*) and (*d*) are 60, 64, 64 and 62, respectively. Consequently, the allowable range $[t_1, t_2]$ equals [148, 150]. Moreover, forward and backward time-switch calculations reveal that FR_{13} = {94, 102, 108, 116, 128, 146, 150, 160, 164} and BR_{13} = {146, 170, 202, 226}. Taken this information into account, the allowable ranges of nodes 5, 6 and 7 of the branch-and-bound tree are given by]-, 146], [150, 150] and [170, -[, respectively. Since $[150, 150] \cap FR_{13} \cap$

$BR_{13} = \varnothing$, the algorithm fathoms node 6 of the search tree. Indeed, all possible solutions to the problem to be found by node 6 will also be found by node 7 of the search tree (section (*b*) of table 1).

Notice also that the algorithm has found feasible project duration for the very first time at node 5 of the tree (with a feasible project duration of 143 time units). The upper bound *ub* (initially set to ∞) is updated as follows: *ub* = 133 and consequently, from this point forward, there is no need to explore nodes with a lower bound larger than 133. The algorithm backtracks when there are no nodes left unexplored at a particular level and continues at the previous level of the search tree. The algorithm continues this way until it returns to the initial node of the search tree.

The optimal solution is reported by the node with the lowest upper bound *ub*, which is found in node 7 of the search tree. The optimal cost of the project with a deadline δ_n = 143 time units amounts to 125.

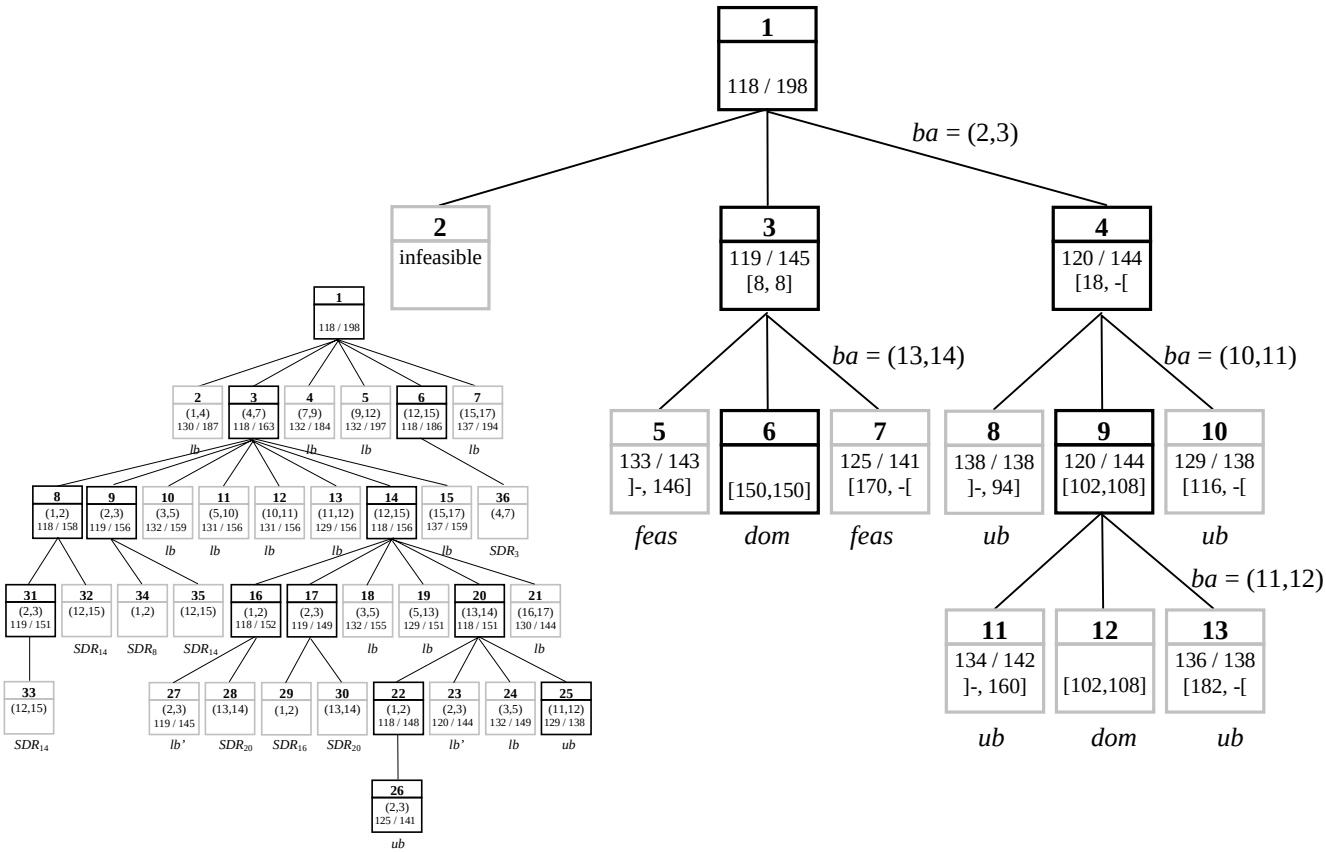

Figure 11. *The branch-and-bound tree of the DVH procedure (left) and the new procedure (right)*

5. COMPUTATIONAL EXPERIENCE

In order to test the efficiency of our new branch-and-bound procedures for the *DTCTPTSC*, we have coded it in Visual C++ version 6.0 under Windows 2000 on a Dell personal computer with a Pentium III, 800 MHz processor. We have used two main problem sets. The first subset is the testset used in Vanhoucke et al. (2002a) and is used to compare our new algorithm with the *VDH* procedure. The second set is extension of this set to larger problem instances and is used to illustrate that our new branch-and-bound procedure can solve problems which seemed to be too complex for the *VDH* procedure. Both testsets are subsets of the dataset used for the computational results of the procedure by Demeulemeester et al. (1998).

5.1. Problem set I.

Table 3 illustrates the characteristics of problem set I and is a copy of the dataset used in Vanhoucke et al. (2002a). This testset is a subset of the problem set generated by Demeulemeester et al. (1998) in which the authors have generated activity-on-the-node networks with different values for the coefficient of network complexity *CNC* with the problem generator *ProGen* (Kolisch et al. (1995)). After this generation, the authors have transformed the networks into an AoA format using the algorithm of Kamburowski et al. (1992). According to the table, we use different settings for the number of modes of each activity, five settings for the deadline of the project and ten settings for the pattern of each network. Using 10 instances for each problem class, we obtain a problem set with 12,000 test instances. The project deadline δ_n is set to a value between the smallest critical path length (all activities at their crash duration) and the largest critical path length (all activities at their normal duration). The project deadline δ_n is then equal to the smallest critical path length exceeded with k times the difference between the largest and the smallest critical path lengths. The settings for the pattern, either *day*, *d&n* or *dnw*, of each activity varies as follows: [% of activities following a *day*-pattern, % of activities following a *d&n*-pattern, % of activities following a *dnw*-pattern].

Table 3. *Parameter settings used to generate the test instances* for the *DTCTPTSC*

ACTIVITY-ON-THE-ARC PROJECT SCHEDULING PROBLEMS GENERATED BY *PROGEN* 12,000 *instances*	
Number of activities (nondummy arcs)	10, 20 or 30
Number of modes	Fixed at 2 or 4 or randomly chosen from the interval [1,3] or [1,7] for the 10-activity networks and fixed at 2 or randomly chosen from the interval [1,3] for the 20- and 30-activity networks
Coefficient of network complexity *CNC*	1.5; 1.8 or 2.1
Pattern [%*day*-pattern, %*d&n*-pattern,%*dnw*-pattern]	[0,0,100], [0,33,66], [0,66,33], [0,100,0], [33,0,66], [33,33,33], [33,66,0], [66,0,33], [66,33, 0], [100,0,0],
Constant *k* for the deadline setting of the project	0, 0.25, 0.50, 0.75 or 1

In table 4 we compare the computational results for the *DVH* procedure with our new branch-and-bound algorithm. To that purpose, we display the average CPU-time in seconds and the average number of nodes in the branch-and-bound tree for solving the *DTCTPTSC* to optimality and we use the same computer as for the *DVH* procedure. The number of nodes in the search tree denotes the number of times a lower bound has been calculated using the procedure of Demeulemeester et al. (1998).

The row label *'all instances'* gives the average results over all 12,000 problem instances and illustrates that our new branch-and-bound procedure clearly outperforms the *DVH* procedure. Our new procedure is, on the average, four times faster and uses seven times less nodes in the branch-and-bound tree. Consequently, the time spent per node is slightly higher in our new procedure. This is the extra time spent on the calculation of the periods t_1 and t_2 and the sets FR_i and BR_i.

In the remaining rows we show more detailed results and illustrate a lot of resemblance between the two procedures.

For the rows labelled '*number of activities*', '*number of modes*', '*CNC*' and '*pattern*', the results correspond with the *DVH* procedure and are extensively described in this paper. Indeed, the larger the number of activities, the more difficult the problem. The negative effect of the number of modes is also quite clear, with the observation that a fixed number of modes is more difficult to solve than the instances where the number of modes is randomly selected. This was already observed for the *DTCTP* in Demeulemeester et al. (1998). The effect is the *CNC* is not so clear in both procedures. It has already been shown extensively in literature that the *CNC* is not a good measure to predict the difficulty of project scheduling problems (see e.g. Elmaghraby and Herroelen (1980), De Reyck and Herroelen (1996), Herroelen and De Reyck (1999) and Vanhoucke et al. (2002b)). The row labelled '*pattern*' illustrates the simplicity of the problem when all arcs follow a

dnw-pattern since then the problem is identical to the *DTCTP*. In all other cases the problem instances are more difficult to solve.

The last results in the row labelled '*deadline*' indicate the effect of the project deadline on the problem complexity and is the most interesting one in our comparison of the two procedures. Instances with a project deadline equal to the largest critical path length are denoted with $k = 1$ and are very easy to solve. The larger the project deadline, the larger the value of k is. Consequently, problem instances solved by the *DVH* procedure for which $k = 0$ are the most difficult. Indeed, the *DVH* procedure starts with a schedule for which all activities are scheduled at their normal mode and iteratively crashes activities during the search of the tree. The closer the deadline to the minimal critical path length, the more activities need to be crashed. Our new branch-and-bound procedure also reveals a negative correlation between the project deadline and the problem complexity, i.e. the larger the project deadline (i.e. the larger k), the more easy the problem. But in contrast with the *DVH* procedure, the new algorithm does not report the largest CPU times for instances with $k = 0$. This is simply because our new procedure does not start with a schedule for which all activities are at their normal duration. Instead, we simply calculate a lower bound at each node of the search tree and split the possible starting time of a chosen branching activity into three parts.

Table 4.

Insert Table 4 About Here

5.2. Problem set II.

Problem set II is similar to problem set I, except for the number of modes for each activity. Instead of restricting the number of modes for the different projects, we now set the modes fixed at 2 or 4 or we randomly choose the modes from the interval [1,3] or [1,7]. Consequently, problem set II contains 18,000 instances and allows us to solve e.g. 30 activity projects for which some activities have up to 7 modes. The *DVH* procedure was not able to solve these large problem instances since the authors conclude that "problem instances with 20 or 30 activities and 4 or more modes are very difficult to solve and therefore, we have to rely on heuristic procedures".

Table 5.

Insert Table 5 About Here

In table 5 we report detailed results of our computational tests. The first row of this table has a similar lay-out as table 4 and needs no extra explanation. Rows 2 and 3 report the average CPU time and the number of created nodes needed to solve the problem instances to optimality. To that

end, we do not use any time limit on the CPU time and we rely on the dominance rule of section 3.2.3 to solve all the instances. In rows 4 and 5 we test the performance of the dominance rule by reporting the average CPU time within a maximum allowed CPU time of 1 minute. Row 4 is similar to row 2, except for the time limit of 60 seconds. In row 5 we solve the same problem set without the use of our dominance rules.

The computational results indicate that instances with 30 activities can be solved in – on the average - less than 7 seconds. Therefore, the algorithm needs to rely 154 times on the procedure of Demeulemeester et al. (1998). The CPU-times of the individual parameters (pattern, number of activities, number of modes, deadline) correspond to the results of table 4. However, it is worth notifying that the *CNC* is positively correlated with the difficulty of the problem instances. The table also reveals that our dominance rule is very efficient: the average CPU time is – on the average – 4.5 times lower with the dominance rule, as observed by comparing rows 4 and 5.

6. CONCLUSIONS

In this paper we presented a new branch-and-bound algorithm for the discrete time/cost trade-off problem with time-switch constraints. The discrete time/cost trade-offs mean that the duration of each activity is a discrete, nonincreasing function of the amount of a single nonrenewable resource. The time-switch constraints impose a specified starting time on the project activities and force them to be inactive during specified time periods.

The problem type is based on a real-life project and has been introduced in the paper by Vanhoucke et al. (2002a). Our new branch-and-bound procedure makes use of a completely new branching strategy and outperforms the previous algorithm. In doing so, we are able to optimally solve larger, real-life project instances. In our future research, we will search for real-life projects in which time/switch constraints are inevitable and extend this problem type with additional features in order to further tighten the gap between the project literature and real-life project management. Solving this problem with heuristic solution methods also belongs to our future intentions.

REFERENCES

Akkan C, Drexl A and Kimms A (2000a). Network decomposition for the discrete time/cost trade-off problem – Part 1: Models and bounding methods. Extended abstracts of the Seventh International Workshop on Project Management and Scheduling. Osnabrück, April 17-19, 29-31.

Akkan C, Drexl A and Kimms A (2000b). Network decomposition for the discrete time/cost trade-off problem – Part 2: Network decomposition and computational results. Extended abstracts of the Seventh International Workshop on Project Management and Scheduling. Osnabrück, April 17-19, 32-34.

Bianco L and Speranza MG (1990). Resource management in project scheduling. Proceedings of the Second International Workshop on Project Management and Scheduling. June 20-22, Compiègne.

Billstein N and Radermacher FJ. (1977). Time-cost optimization. *Methods of Operations Research*, Vol. 27, 274-294.

Crowston W. (1970). Network reduction and solution. *Operations Research Quarterly*, Vol. 21, 435-450.

Crowston W and Thompson GL. (1967). Decision CPM: A method for simultaneous planning, scheduling and control of projects. *Operations Research*, Vol. 15, 407-426.

De P, Dunne EJ, Ghosh JB and Wells CE. (1995). The discrete time/cost trade-off problem revisited. *European Journal of Operational Research*, Vol. 81, 225-238.

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

Demeulemeester E, De Reyck B, Foubert B, Herroelen W and Vanhoucke M. (1998). New computational results for the discrete time/cost trade-off problem in project networks. *Journal of the Operational Research Society*, Vol. 49, 1153-1163.

Demeulemeester E, Elmaghraby SE and Herroelen W. (1996). Optimal procedures for the discrete time/cost trade-off problem in project networks. *European Journal of Operational Research*, Vol. 88, 50-68.

De Reyck, B. and Herroelen, W. (1996). On the use of the complexity index as a measure of complexity in activity networks. *European Journal of Operational Research*, 91, 347-366.

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. (1980). On the measurement of complexity in activity networks. *European Journal of Operational Research*, 5, 223-234.

Elmaghraby SE and Kamburowski J. (1992). The analysis of activity networks under generalized precedence relations. *Management Science*, Vol. 38, 1245-1263.

Elmaghraby SE and Salem A. (1980a). Optimal project compression under convex cost functions I: Quadratic cost with continuous derivative. OR Technical Report 158, North Carolina State University at Raleigh.

Elmaghraby SE and Salem A. (1980b). Optimal project compression under convex cost functions II. OR Technical Report 157, North Carolina State University at Raleigh.

Elmaghraby SE and Salem A. (1981). Optimal linear approximation in project compression. OR Technical Report 171, North Carolina State University at Raleigh.

Falk JE and Horowitz JL. (1972). Critical path problems with concave cost-time curves. *Management Science*, Vol. 19, 446-455.

Ford LR and Fulkerson DR. (1962). Flows in networks. Princeton University Press: New Jersey.

Fulkerson DR. (1961). A network flow computation for project cost curves. *Management Science*, Vol. 7, 167-178.

Goyal SK. (1975). A note on the paper: A simple CPM time/cost trade-off algorithm. *Management Science*, Vol. 21, 718-722.

Herroelen, W. and De Reyck, B. (1999). Phase transitions in project scheduling. *Journal of Operational Research Society*, 50, 148-156.

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.

Hindelang TJ and Muth JF. (1979). A dynamic programming algorithm for decision CPM networks *Operations Research*, Vol. 27, 225-241.

Kamburowski J, Michael DJ and Stallmann MFM. (1992). Optimal construction of project activity networks. Proceedings of the 1992 Annual Meeting of the Decision Sciences Institute. San Francisco, 1424-1426.

Kapur KC. (1973). An algorithm for the project cost/duration analysis problem with quadratic and convex cost functions. *IIE Transactions*, Vol. 5, 314-322.

Kelley JE. (1961). Critical path planning and scheduling: Mathematical basis. *Operations Research*, Vol. 9, 296-320.

Kelley JE and Walker MR. (1959). Critical path planning and scheduling: An introduction. Mauchly Associates, Ambler, PA.

Kolisch R, Sprecher A and Drexl A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, Vol. 41, 1693-1703.

Lamberson LR and Hocking RR. (1970). Optimum time compression in project scheduling. *Management Science*, Vol. 16, B-597-B-606.

Moder JJ, Phillips CR and Davis EW. (1983). Project Management with CPM, PERT and precedence diagramming. Van Nostrand Reinhold Company, Third Edition.

Patterson JH and Harvey RT. (1979). An implicit enumeration algorithm for the time/cost trade-off problem in project network analysis. *Foundations of Control Engineering*, Vol. 6, 107-117.

Robinson DR. (1975). A dynamic programming solution to cost/time trade-off for CPM. *Management Science*, Vol. 22, 158-166.

Siemens N. (1971). A simple CPM time/cost trade-off algorithm. *Management Science*, Vol. 17, B-354-363.

Siemens N and Gooding C. (1975). Reducing project duration at minimum cost: a time/cost trade-off algorithm. *OMEGA*, Vol. 3, 569-581.

Skutella M. (1998). Approximation algorithms for the discrete time-cost tradeoff problem. *Mathematics of Operations Research*, Vol. 23, 909-929.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W. (2002a) Discrete time/cost trade-offs in project scheduling with time-switch constraints. *Journal of the Operational Research Society*, Vol. 53, 1-11.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W. (2002b) A random network generator for activity-on-the-node networks. *Journal of Scheduling*, to appear.

Vercellis C. (1990). Multi-project scheduling with time-resource-cost trade-offs: a tactical model. Proceedings of the Second International Workshop on Project Management and Scheduling, June 20-22, Compiègne.

Wiest JD and Levy FK. (1977). A management guide to PERT/CPM: with GERT/PDM/DCPM and other networks. Prentice-Hall, Inc.

Yang HH and Chen YL. (2000). Finding the critical path in an activity network with time-switch constraints. *European Journal of Operational Research*, Vol. 120, 603-613.

APPENDIX

In this appendix we explain table 1 of section 3.2.3 by means of three simple projects illustrations. In each of the cases below, we assume that the start of the project equals Wednesday (time zero) with a project deadline of 16 work periods (see figure 12). Moreover, we always assume that the lower bound calculation has reported a schedule in which activity (1,2) takes 8 work periods and in which activity (2,3) is the selected branching activity (with a time range $[t_1, t_2] = [8, 8]$ in all cases).

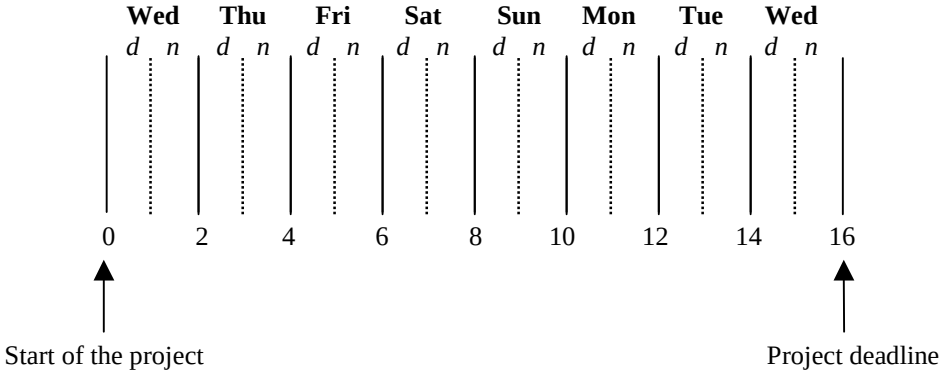

Figure 12. *Time window for the project example*

Case (*a*)

It is easy to see on figure 13 that $FR_2 = \{2, 12\}$ and $BR_2 = \{4, 10, 14\}$. Since $[8, 8] \cap FR_2 = \varnothing$ and $[8, 8] \cap BR_2 = \varnothing$, we follow case (a) of table 1. Therefore, we fathom node 3 of the branch-and-bound tree and use]-, 2] and [12, -[for node 2. Indeed, it is not necessary to explore node 2 with a time range [10, -[since then we have an overlap between node 1 and node 2. Consequently, the possibility that activity (2, 3) is scheduled at a duration of 6 can be excluded of node 2 since this in incorporated in the exploration of node 1.

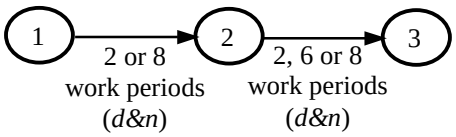

Figure 13. *Project example for case (a)*

Case (*b*)

In the example network of figure 14, $FR_2 = \{2, 8, 12\}$ and $BR_2 = \{4, 10, 14\}$ and $[8, 8] \cap FR_2 \neq \emptyset$ and $[8, 8] \cap BR_2 = \emptyset$. According to table 1, we can fathom node 3 and use time ranges of nodes 1 and 2 of]-, 2] and [10, -[, respectively. In this case, it is really necessary to explore node 2 with a time range [10, -[instead of [12, -[since otherwise, we exclude the possible solution in which activity (1, 2) is scheduled at duration 8 and activity (2, 3) is scheduled at duration 6.

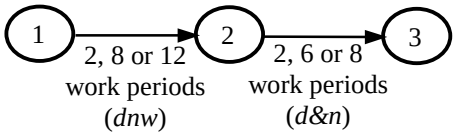

Figure 14. *Project example for case (b)*

Case (*c*)

The example network of figure 15 reveals that $FR_2 = \{2, 12\}$ and $BR_2 = \{8, 10, 12, 14\}$ and $[8, 8] \cap FR_2 = \emptyset$ and $[8, 8] \cap BR_2 \neq \emptyset$. According to table 1, we can fathom node 3 and use time ranges of nodes 1 and 2 of]-, 2] and [12, -[, respectively. For a similar reason of case (a), we do not need to investigate node 2 with range [10, -[, since this has been incorporated in node 1.

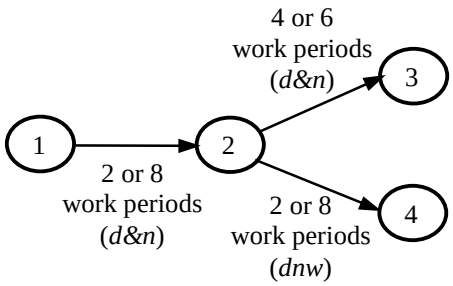

Figure 15. *Project example for case (c)*

Case (*d*)

In this case, we do not fathom node 3.

TABLE 4
A comparison of the average CPU-time and number of created nodes in the branch-and-bound tree between our new branch-and-bound procedure and the algorithm by Vanhoucke et al.

	Average CPU time (seconds)		**Average number of nodes in the search tree**	
	VDH (2002)	***Current paper***	***VDH (2002)***	***Current paper***
All instances	0.119	0.027	88.718	11.242
Number of activities	0.006	0.003	25.325	2.578
10	0.016	0.010	26.899	8.366
20	0.449	0.089	277.321	31.444
30				
Number of modes (10 activities)				
2	0.001	0.003	1.701	1.953
4	0.022	0.005	95.437	4.353
[1,3]	0.001	0.003	1.442	2.142
[1,7]	0.001	0.003	2.718	1.865
(20 and 30 activities)	0.415	0.080	251.861	24.190
2	0.051	0.021	52.359	13.728
[1,3]				
CNC				
1.5	0.081	0.024	98.447	11.367
1.8	0.146	0.021	92.189	9.004
2.1	0.133	0.036	75.516	11.935
Pattern				
[0,0,100]	0.002	0.004	1.000	1.000
[0,33,66]	0.092	0.006	50.373	2.669
[0,66,33]	0.324	0.026	146.487	12.913
[0,100,0]	0.274	0.066	204.304	28.497
[33,0,66]	0.021	0.005	22.430	2.059
[33,33,33]	0.061	0.020	93.446	7.520
[33,66,0]	0.077	0.060	92.784	20.081
[66,0,33]	0.051	0.016	50.287	6.784
[66,33, 0]	0.133	0.032	115.203	13.763
[100,0,0]	0.161	0.035	111.858	12.403
Deadline				
$k = 0$	0.529	0.040	408.275	12.395
$k = 0.25$	0.062	0.059	28.625	20.404
$k = 0.50$	0.004	0.023	5.451	12.285
$k = 0.75$	0.002	0.009	1.235	7.760
$k = 1$	0.001	0.003	1.000	1.000

(2002a) for problem set I

TABLE 5.

Computational results for our new branch-and-bound procedure for problem set II

	Average CPU time (seconds) (no time limit)	**Average number of nodes in the search tree** (no time limit)	**Average CPU time (seconds)** (time limit 1 minute)	**Average CPU time (seconds)** (time limit 1 minute)
	With dominance rule	***With dominance rule***	***With dominance rule***	***Without dominance rule***
All instances	2.209	60.189	0.942	4.339
Number of activities				
10	0.003	2.579	0.003	0.004
20	0.089	23.466	0.089	1.507
30	6.535	154.524	2.733	11.506
Number of modes				
2	0.044	17.860	0.044	2.195
4	5.133	132.345	2.283	7.676
[1,3]	0.012	10.050	0.012	1.825
[1,7]	3.646	80.503	1.427	5.660
CNC				
1.5	1.032	45.555	0.577	3.647
1.8	1.695	53.295	0.781	3.742
2.1	3.901	81.718	1.467	5.628
Pattern				
[0,0,100]	0.007	1.000	0.007	0.007
[0,33,66]	0.110	5.352	0.110	0.221
[0,66,33]	2.954	74.144	1.233	4.573
[0,100,0]	7.773	176.624	2.595	10.666
[33,0,66]	0.051	3.958	0.051	0.116
[33,33,33]	0.594	28.577	0.519	2.566
[33,66,0]	2.295	99.119	1.478	8.292
[66,0,33]	1.930	35.023	0.613	3.189
[66,33, 0]	3.392	98.836	1.513	7.103
[100,0,0]	2.984	79.260	1.298	6.656
Deadline				
$k = 0$	1.947	49.147	1.185	7.330
$k = 0.25$	6.992	138.296	2.602	7.728
$k = 0.50$	2.046	86.656	0.861	4.657
$k = 0.75$	0.056	25.848	0.056	1.976
$k = 1$	0.003	1.000	0.003	0.003