Network decomposition-based benchmark results for the discrete time–cost tradeoff problem

Can Akkan [a,*], Andreas Drexl [b], Alf Kimms [c]

[a] Graduate School of Management, Sabanci University, Orhanli, Tuzla, 34956 Istanbul, Turkey
[b] Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24118 Kiel, Germany
[c] Fakultät für Wirtschaftswissenschaften, Technical University Bergakademie Freiberg, Lessingstr. 45, 09596 Freiberg, Germany

Abstract

In project management, the project duration can often be compressed by accelerating some of its activities at an additional expense. This is the so-called time–cost tradeoff problem which has been extensively studied in the past. However, the discrete version of the problem which is of great practical relevance, did not receive much attention so far. Given a set of modes (time–cost pairs) for each activity, the objective of the discrete time–cost tradeoff problem is to select a mode for each activity so that the total cost is minimized while meeting a given project deadline.

The discrete time–cost tradeoff problem is a strongly $\mathcal{NP}$-hard optimization problem for general activity networks. In terms of what current state-of-art algorithms can do, instances with (depending on the structure of the network and the number of processing alternatives per activity) no more than 20–50 activities can be solved to optimality in reasonable amount of time. Hence, heuristics must be employed to solve larger instances. To evaluate such heuristics, lower bounds are needed. This paper provides lower and upper bounds using column generation techniques based on "network decomposition". Furthermore, a computational study is provided to demonstrate that the presented bounds are tight and that large and hard instances can be solved in short run-time.

Keywords: Project scheduling; Activity network; Time–cost tradeoff; Network decomposition; Column generation

1. Introduction

In managing a project, there is often the option of applying additional funds to reduce the processing time of some activities. Models developed for tackling this tradeoff are known as time–cost tradeoff models. There are two possible objectives for these models. One could either try to find the minimum cost activity processing times while meeting a given deadline (the *deadline problem*) or minimize the duration of the

* Corresponding author.
E-mail addresses: canakkan@sabanciuniv.edu (C. Akkan), drexl@bwl.uni-kiel.de (A. Drexl), alf.kimms@bwl.tu-freiberg.de (A. Kimms).

entire project meeting a given budget constraint (the *budget problem*). Also it might be of interest to compute the entire project cost curve. In this paper we present network decomposition methods for the deadline problem.

The discrete time–cost tradeoff problem (DTCTP) can be stated as follows: Given a set of modes (time–cost pairs) for each activity, the objective is to select a mode for each activity so that the total cost is minimized while meeting a given project deadline. The DTCTP is a strongly $\mathcal{NP}$-hard optimization problem for general activity networks [7]. The purpose of this paper is to introduce lower bounds and upper bounds which are computed through network decomposition techniques. The purpose of network decomposition is to obtain a set of subnetworks that can be solved quickly (by a general purpose MIP solver).

The modeler has to choose one of two types of networks: activity-on-arc (AoA) or activity-on-node (AoN). So far the relevant literature most often is based on the AoA representation although it is well known that the AoA representation is inferior to the AoN representation. In fact, Krishnamoorthy and Deo [17] (see also [24]) have shown that it is an $\mathcal{NP}$-complete optimization problem to compute the minimum number of dummy arcs necessary to represent the given precedence constraints between the activities when using the AoA format while no dummy arcs are needed at all for AoN networks. Hence, we use the AoN representation throughout the paper.

The paper is organized as follows: In Section 2, the notation is introduced and the problem setting is discussed. Related work is reviewed in Section 3. Section 4 contains a mathematical programming model formulation for the DTCTP. In Section 5, problem insight is used to obtain cuts which tighten the LP-relaxation based lower bound. Preprocessing techniques are discussed in Section 6. In Section 7, the idea of network decomposition is explained. Section 8 presents a column generation procedure that employs the network decomposition idea to compute lower bounds. Feasible solutions are derived in Section 9. A report on in-depth computational studies is provided in Section 10. Finally, concluding remarks in Section 11 finish the paper.

2. Problem setting

The DTCTP was introduced by Harvey and Patterson [13] and Hindelang and Muth [14]. It can formally be described as follows: It consists of a set $V = \{0, 1, \ldots, n, n+1\}$ of activities, a set $E \subseteq V \times V$ of immediate precedence constraints among the activities, and, for each activity j, an index set $M_j = \{\underline{m}_j, \ldots, \bar{m}_j\}$ of possible modes. A mode $m \in M_j$ is characterized by a processing time $p_{jm} \in \mathbb{N}_0$ and a cost $c_{jm} \in \mathbb{R}_{\geqslant 0}$. Without loss of generality, we assume that, for each $j \in V$, $m < m'$ implies $p_{jm} > p_{jm'}$ and $c_{jm} < c_{jm'}$. Furthermore, it is not restrictive to assume that activity 0 is the unique source node in the network $G = (V, E)$ and that activity $n+1$ is the unique sink node.

A realization of the project is an assignment of modes to all activities $j \in V$. Let $m_j \in M_j$ be the mode assigned to activity j. Then, the total cost of the realization is given by $\sum_{j \in V} c_{jm_j}$. The makespan of the realization is the makespan of the earliest start schedule of the project when activity j has processing time p_{jm_j}.

Limiting either cost or time, we obtain two related optimization problems:

(i) For a given deadline T on the makespan, find a realization with a makespan less than or equal to T that minimizes the total cost of the realization (*deadline problem*).
(ii) For a given budget $B \geqslant 0$, find a realization with a total cost that is less than or equal to B that minimizes the makespan of the realization (*budget problem*).

The emphasis of this paper is on the deadline problem.

Solving the deadline problem for all possible deadlines T leads to the function $B_{opt}(T)$ giving the minimum cost as a function of the deadline T (the so-called project cost curve). Analogously, solving the budget problem for all possible budgets B leads to the function $T_{opt}(B)$ giving the minimum makespan as a function of the budget B.

3. Related work

The importance of the time–cost tradeoff problem was brought to the attention of the research community almost 40 years ago [12,15,16], but still, due to its inherent computational complexity, research in the DTCTP is rather sparse.

3.1. Exact algorithms

Due to the practical importance of the DTCTP, many exponential time exact algorithms have been proposed. Early examples are dynamic programming approaches by Hindelang and Muth [14] and Robinson [20], and an enumeration algorithm by Harvey and Patterson [13].

The currently best known algorithms still rely on dynamic programming, but exploit in addition the decomposition structure of the underlying network. The decomposition that facilitates the computation is known as modular decomposition or substitution decomposition and has many applications in network and other combinatorial optimization problems (see the comprehensive article by Möhring and Radermacher [18]).

Its usefulness for the time–cost tradeoff problem was first observed by Frank et al. [11] and Rothfarb et al. [21] for the special case of series–parallel decompositions. The general decomposition theorem that involves arbitrary modules is due to Billstein and Radermacher [5].

Because of the modular decomposition, the project cost curve needs only to be evaluated for certain indecomposable subnetworks (the factors in a composition series) of the original network. This is done by "transforming" such an indecomposable network to a series–parallel network and then performing the "easy" calculations for the series–parallel case. The transformation into a series–parallel network successively identifies certain nodes for "duplication". Any such duplication transforms the network "closer" to a series–parallel one, but increases the computation time by a multiplicative factor.

This idea seems to be due to Robinson [20] and has been further developed by Bein et al. [4], De et al. [6] and Elmaghraby [10].

Demeulemeester et al. [9] provide the first implementation of this approach. They implement two strategies for finding the nodes for duplication: The first follows the theory of Bein et al. [4], which results in the minimum number of duplications required, while the second tries to minimize the number of realizations that have to be considered during the algorithm. Demeulemeester et al. [9] report on computational experience for networks with up to 45 activities without identifying a clear winner between the two strategies.

The crucial parameter in the theoretical run-time analysis of this algorithmic approach is the minimum number of node duplications needed to transform an AoA network into a series–parallel network. Bein et al. [4] refer to it as the reduction complexity of the AoA network (also known as the complexity index, CI, of an AoA network). It provides a measure for the "distance" of the given network from being series–parallel.

3.2. Lower bounds

To the best of our knowledge, only one non-trivial lower bounding procedure based on Lagrangean relaxation is available for the DTCTP [1]. The Lagrangean relaxation is applied on an induced AoA

network that is created by adding new nodes so that the resulting network is made up of a set of series networks. These series networks are linked by constraints (which are later relaxed by Lagrangean multipliers) that require the equivalent nodes among them to have the same occurrence times. The Lagrangean subproblem is solved by dynamic programming. Unfortunately, for instances with large time-windows for occurrence times of nodes, the CPU-time required for the solution of the dynamic programming problems becomes prohibitively large.

3.3. Approximation algorithms

Recently, the approximation behavior of the DTCTP has been analyzed by Skutella [22]. He first presents a polynomial reduction to the case where every activity has at most two processing times, and one of them is zero. For such a DTCTP, Skutella defines a "natural" linear relaxation. One then first solves the linear relaxation. This solution is then "rounded" to a solution of the original problem by rounding the processing times of some activities. Linear relaxation and rounding are key means in order to come up with different polynomial approximation schemes with guaranteed performance for both the deadline and the budget problem.

Deineko and Woeginger [8] prove an in-approximability result with respect to polynomial time bicriteria approximation algorithms for this problem.

3.4. Heuristic algorithms

Akkan [1] developed a Lagrangean relaxation-based heuristic for AoA networks which borrows from the ideas developed for the computation of lower bounds. Two local search algorithms for AoA networks have been developed by Akkan [2]. Both utilize a steepest descent algorithm for finding a local optimum given a starting solution, which is constructed using a path heuristic. The first local search algorithm is based on random restarts while the second uses random kicks. The steepest descent algorithm is based on two local moves from a given solution: The first one identifies for a given node the minimum decrease in the starting time required to have an outgoing activity use a longer (cheaper) mode, while keeping the starting times of all other nodes unchanged. Similarly, the second move identifies the minimum required increase in the starting time of a given node to have an incoming activity use a longer mode. These heuristics yield solutions within 1% of the optimum for (small) test problems from the literature within fractions of seconds.

3.5. Generalization

An important generalization of the DTCTP is the so-called multi-mode resource-constrained project scheduling problem. There, M_j is used in a similar way to model resource–resource and time–resource tradeoffs. Resources are either renewable or non-renewable, for details, see for instance [23].

4. Model formulation

A mathematical programming model of the deadline variant of the DTCTP can be stated using the following additional notation. Let EC_j (ES_j) denote the earliest completion (start) time and LC_j denote the latest completion time of activity j. Note, EC_j and LC_j can be calculated easily using the shortest processing times of the activities. We use C_j to denote the completion time of activity j (a decision variable). Furthermore, we use a binary decision variable x_{jm} that takes the value one, if mode m is chosen for activity j. Otherwise, its value is zero. The MIP model formulation reads as follows:

$$\min \quad \sum_{j \in V} \sum_{m \in M_j} c_{jm} x_{jm}, \tag{1}$$

$$\text{s.t.} \quad \sum_{m \in M_j} x_{jm} = 1, \quad j \in V, \tag{2}$$

$$C_j - C_i - \sum_{m \in M_j} p_{jm} x_{jm} \geqslant 0, \quad (i,j) \in E, \tag{3}$$

$$C_{n+1} \leqslant T, \tag{4}$$

$$C_0 - \sum_{m \in M_0} p_{0m} x_{0m} \geqslant 0, \tag{5}$$

$$C_j \geqslant 0, \quad j \in V, \tag{6}$$

$$x_{jm} \in \{0,1\}, \quad j \in V, \ m \in M_j. \tag{7}$$

While minimizing the total cost (1), a unique mode must be assigned to each activity (2), precedence constraints must not be violated (3), the deadline must be met (4), and the project may not start before time zero (5).

5. Cuts

Solving the LP-relaxation of the model formulation gives a lower bound. To tighten the lower bound, we add additional constraints (cuts). Formally, we consider the transitive closure E^* of the precedence relation E, i.e. a pair of activities (i,j) is a member of the set E^*, if and only if j is a successor (not necessarily an immediate one) of activity i.

For every pair $(i,j) \in E^*$ and all combinations of $m_i \in M_i$, and $m_j \in M_j$, we test if

$$\mathrm{LC}_j - \mathrm{ES}_i < p_{im_i} + p_{jm_j}$$

is true. If this test succeeds,

$$x_{im_i} + x_{jm_j} \leqslant 1 \tag{8}$$

is a possible cut. Note that if $x_{m_i} + x_{jm_j} \leqslant 1$ is a cut, every combination of modes

$$\{\underline{m}_i, \ldots, m_i\} \times \{\underline{m}_j, \ldots, m_j\}$$

defines a cut as well, because a lower mode index means a longer processing time. This may result in a huge number of cuts, but fortunately we can do a bit better, because, if $x_{im_i} + x_{jm_j} \leqslant 1$ is a cut,

$$\sum_{m=\underline{m}_i}^{m_i} x_{im} + \sum_{m=\underline{m}_j}^{m_j} x_{jm} \leqslant 1 \tag{9}$$

is an even stronger cut which is used instead.

It is easy to see that (9) is a stronger cut than the system of $(m_i - \underline{m}_i + 1) \times (m_j - \underline{m}_j + 1)$ cuts implied by (8), because every solution that fulfills (9) also fulfills every cut implied by (8), because the variables x_{jm} are defined to be non-negative. The reverse is not true. Of course, only non-dominated cuts need to be used.

It should be remarked that what has been defined for pairs of activities $(i,j) \in E^*$ can be extended for arbitrary tuples of activities. But, there is a tradeoff between tight lower bounds and the computational effort for computing the cuts. For this reason, all our computational tests are based on using cuts derived from considering pairs of activities only.

Note that each of the cuts (9) can be identified by a four-tuple of the form (i, m_i, j, m_j) for short which will ease the notation later on. Furthermore, we will use $\mathscr{C}$ to denote the set of all such four-tuples representing non-dominated cuts.

6. Mode elimination

Before turning to solve a particular instance, it is often worth to have a look at that instance to see, if it can be simplified. In this section we will discuss techniques which help to eliminate some modes. As before, let

$$M_j = \{\underline{m}_j, \ldots, \overline{m}_j\}$$

be the index set of modes for activity j which have not been eliminated yet. This set will be further reduced during preprocessing, and for the sake of notational convenience M_j will be used to denote the updated set. The index $\underline{m}_j$ will be used to identify the mode with the longest processing time (the "longest" mode) in the current set M_j, and the index $\bar{m}_j$ will be used to identify the mode with the shortest processing time (the "shortest" mode) in the current set M_j.

6.1. Eliminating "short" modes

Consider all source-to-sink paths where such a path l is defined to be

$$Q_l = \{0 = j(l0), j(l1), \ldots, j(lq_l) = n+1\} \subseteq V.$$

L is the number of all such paths and $(j(l(k-1)), j(lk)) \in E$ for all $k = 1, \ldots, q_l$, i.e. a path Q_l, or l for short, is represented by the set of activities on this path.

For each path l, it is easy to compute the longest possible path length

$$\mathscr{L}(Q_l) = \sum_{i=0}^{q_l} p_j(li)\underline{m}_{j(li)} \tag{10}$$

which is simply the sum of longest processing times of the activities on that path.

Now, determine all paths which have a longest possible path length that exceeds the deadline T. Note that there is at least one such path, because otherwise the instance would be trivial and all activities could be scheduled in their "longest" mode. Let

$$\mathscr{Q} = \bigcup_{l=1,\ldots,L \mid \mathscr{L}(Q_l) > T} l$$

denote the set of activities which lie on at least one such T-exceeding path.

Actually, we are not interested in the set of activities in $\mathscr{Q}$, but in the set of activities which are not in $\mathscr{Q}$, because these activities can be scheduled using their "longest" mode no matter what mode assignment is chosen for other activities. Formally, we have that for all

$$j \in V \setminus \mathscr{Q},$$

where

$$|M_j| > 1,$$

the index set of modes can be reduced to a single element, i.e.

$$M_j = \{\underline{m}_j\}.$$

It is interesting to note that calculating the earliest start times ES_j and the latest completion times LC_j for the activities using the "shortest" modes in the current sets M_j, the elimination of "short" modes may reduce the time interval $[ES_j; LC_j]$ for some activities. This effect is of interest, because the elimination of "long" modes uses these time intervals.

6.2. Eliminating "long" modes

Now we consider the time interval $[ES_j; LC_j]$ for each of the activities where the earliest start times ES_j and the latest completion times LC_j are computed using the "shortest" modes in the current sets M_j.

Given such a time interval, it is clear that for every $j \in V$ for which a mode $m \in M_j$ exists such that

$$p_{jm} > LC_j - ES_j$$

holds, the mode m can be eliminated, i.e.

$$M_j = M_j \setminus \{m\}.$$

Note that eliminating "long" modes of an activity j reduces the longest possible length (see (10)) of all paths l with $j \in Q_l$ and thus may reduce the set $\mathcal{Q}$ that is defined to eliminate "short" modes. Hence, eliminating "long" modes affects the elimination of "short" modes.

The elimination of "short" and "long" modes iterates until the set of modes cannot be reduced further.

7. Network decomposition

The procedure we propose for computing lower bounds is based on a decomposition of the network which is a partition of the set of activities. For a given partition, we use H to denote the index set of the subsets of the activities, and $V^h \subseteq V$ to denote the subset $h \in H$ of the activities. Decomposing a network can now be specified more formally. It means to construct $|H|$ sets of activities such that $\cup_{h \in H} V^h = V$ and $V^h \cap V^{h'} = \emptyset$, if $h \neq h'$. Given a partition of the set of activities, we will use $E^h \subseteq V^h \times V^h \subseteq E$ to denote the set of all precedence constraints within subnetwork G^h, or h for short. On the other hand, $\overline{E} = E \setminus \cup_{h \in H} E^h$ is used to denote the set of precedence constraints among activities which do not belong to the same subnetwork.

Decomposing a network is guided by several insights into the problem. First of all, it is reasonable to have connected subnetworks, because otherwise a subnetwork could be further decomposed while giving the same arc set $\overline{E}$.

Second, while having a model formulation like (1)–(7) in mind, it is reasonable to assume that the computational effort for solving an instance of such a model highly depends on the number of binary decision variables in the model. Hence, a plausible measure for the size of a subnetwork is related to the number of binary decision variables in the aforementioned model formulation.

Third, activities which have just one single mode do not contribute to the computational effort, because the mode assignment for such activities is trivial. Hence, it makes sense to use, for each activity, the number of modes minus one to compute the size of a subnetwork. Formally, a subnetwork consisting of the activities in the set V^h is defined to have a size equal to $\sum_{j \in V^h} (|M_j| - 1)$.

In summary, our aim is to decompose a network in such a way that we get connected subnetworks with

$$\sum_{j \in V^h} (|M_j| - 1) \leqslant b$$

for each subnetwork h where b is a parameter that controls the size of the subnetworks. The number $|H|$ of subnetworks will be the result of the network decomposition rather than an input parameter for decomposing a network.

In some preliminary computational tests not reported here in detail, we have observed that increasing b improves the lower bounds slightly, but requires exponentially increasing run-time. Given a value b, there is a universe of procedures that can be applied for creating subnetworks, but we have found that the way a network is decomposed, once b is fixed, seems to have a minor impact on the performance only. Thus, we confine our discussion to a single method for creating subnetworks which is the procedure that we have eventually used in our tests.

We initially assume that every activity makes up an individual subnetwork, i.e. initially we assume to have $|H| = n + 2$ subnetworks which means that $\overline{E} = E$. Let Γ be the graph that represents the subnetworks as nodes and some precedence constraints among the subnetworks. Initially, Γ equals G.

The basic operation is a merge operation applied to two subnetworks which are connected by at least one arc in Γ. This operation is illustrated in Fig. 1. Suppose, two nodes (subnetworks) i and j in Γ should be merged. Then, these nodes may have one or more immediate predecessors (i^- and j^+, respectively) in Γ and one or more immediate successors (i^- and j^-, respectively) in Γ. Merging i and j means to contract Γ in such a way that the arc that connects i and j is deleted and i and j are now represented by one common node that represents a subnetwork which combines subnetworks i and j. If i and j share a common immediate predecessor or a common immediate successor, parallel arcs would appear, but we eliminate these and want to have unique arcs only.

We proceed iteratively by merging subnetworks which are connected by at least one arc out of the current set $\overline{E}$. While doing so, some issues must be taken into account: First, the resulting subnetwork must not have a size exceeding b. And second, choosing two subnetworks for a merge operation, merging them, and starting the procedure all over again bears the risk that a few subnetworks grow larger and larger (up to b) while some other fairly small subnetworks remain. Hence, our procedure is a bit more complicated in the sense that within each iteration several pairs of subnetworks may be merged. We make sure that within each iteration each subnetwork participates in at most one merge operation, because this allows us to check easily, if performing the merge operations does not violate the restriction imposed by the parameter b; all that need to be done is to add the size of the two subnetworks to which a merge operation should be applied. The pair of subnetworks to be merged is determined in the following manner. Find the largest subnetwork (if there are ties choose any of it) and find the largest subnetwork that is connected to it such that the sum of the sizes of these two does not exceed b. Note, such a neighbor subnetwork may not exist which simply means that the largest subnetwork cannot be augmented any further. If such a pair of subnetworks can be found, mark them for performing a merge operation later on. Then, find the largest subnetwork which may be merged with another one and proceed likewise. If no further pairs of subnetworks can be identified, merge them, and start another iteration working on the contracted network. The procedure stops, if an iteration is reached where no subnetworks can be merged any more.

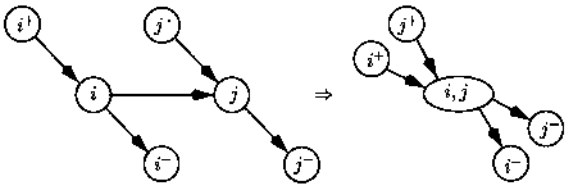

Fig. 1. Merging two subnetworks.

8. Column generation

Recall that, from the preprocessed network, we can compute the earliest start ES_j and, given T, the latest completion time LC_j for each activity $j \in V$ by using the shortest processing times for calculation.

Furthermore, suppose that, for a given instance, we have decomposed the network so that each activity j belongs to one subnetwork $h(j) \in H$. Note that the set $\mathscr{C}$ of cuts of the form (9) can be partitioned such that

$$\mathscr{C} = \mathscr{C}_{\overline{E}} \cup \bigcup_{h \in H} \mathscr{C}_{E^h},$$

where $(i, m_i, j, m_j) \in \mathscr{C}_{\overline{E}}$ if and only if $h(i) \neq h(j)$ and $(i, m_i, j, m_j) \in \mathscr{C}_{E^h}$ if and only if $h(i) = h(j) = h$.

8.1. Master problem

Now, consider any subnetwork h. At least theoretically, we can enumerate all mode assignments and all schedules for subnetwork h. Note that the number of such schedules is exponentially large and therefore we need a more sophisticated procedure. Details of how to do this are not important at this point and we will discuss them in Section 8.2.

Let S^h be the index set of all feasible mode assignments and schedules for subnetwork h where feasibility basically means that the precedence constraints E^h are respected and that each activity is scheduled within its time window $[ES_j; LC_j]$. Each solution $s \in S^h$ can be represented by completion times C_{js} and a binary vector of mode assignments x_{jms} for each activity $j \in V^h$. The cost of solution s for the subnetwork h is defined to be

$$c_{hs} = \sum_{j \in V^h} \sum_{m \in M_j} c_{jm} x_{jms}. \tag{11}$$

The overall DTCTP can now be specified as being the problem of selecting exactly one solution s for each subnetwork h such that the precedence constraints $\overline{E}$ are taken into account. Formally, a binary decision variable z_{hs} can be used to model that selection where a value one indicates that solution s for subnetwork h is selected, and a value zero indicates that it is not selected. This leads to the following model formulation for the DTCTP:

$$\min \quad \sum_{h \in H} \sum_{s \in S^h} c_{hs} z_{hs} \tag{12}$$

$$\text{s.t.} \quad \sum_{s \in S^h} z_{hs} = 1, \quad h \in H, \tag{13}$$

$$\sum_{s \in S^{h(j)}} C_{js} z_{h(j)s} - \sum_{s \in S^{h(i)}} C_{is} z_{h(i)s} - \sum_{s \in S^{h(j)}} \sum_{m \in M_j} p_{jm} x_{jms} z_{h(j)s} \geqslant 0, \quad (i,j) \in \overline{E}, \tag{14}$$

$$-\sum_{s \in S^{h(i)}} \sum_{m=\underline{m}_i}^{m_i} x_{ims} z_{h(i)s} - \sum_{s \in S^{h(j)}} \sum_{m \in \underline{m}_j}^{m_j} x_{jms} z_{h(j)s} \geqslant -1, \quad (i, m_i, j, m_j) \in \mathscr{C}_{\overline{E}}, \tag{15}$$

$$z_{hs} \in \{0,1\} \quad h \in H, \ s \in S^h. \tag{16}$$

The objective (12) is to minimize the total cost. Due to (13) exactly one solution must be selected for each subnetwork h. The constraints (14) guarantee that solutions are selected in such a way that the precedence constraints $\overline{E}$ are taken into account. The cuts (15) are the ones that are defined in (9). Note that (15) is redundant in the presence of a binary domain for the decision variables (16).

The problem with this model reformulation is that there are too many variables, i.e. an exponential number, for allowing us to solve an instance of this model optimally. But, this model helps to compute a lower bound, because, as we will see, we can optimally solve the LP-relaxation of this model with column generation techniques. Considering the LP-relaxation means to replace (16) with

$$z_{hs} \geqslant 0, \quad h \in H, \ s \in S^h.$$

Constraints of the form

$$z_{hs} \leqslant 1 \quad h \in H, \ s \in S^h,$$

need not be considered, because of (13).

To apply column generation, we start with exactly one column for each subnetwork initially. These columns represent an earliest start schedule using "shortest" modes. Formally, this is to say that, for each activity j, $C_{j1} = \mathrm{EC}_j$, $x_{j\overline{m}_j 1} = 1$, and $x_{jm1} = 0$ for all $m \in M_j \setminus \{\overline{m}_j\}$. The objective function coefficients for these columns are determined by (11). It is important to note that the initial restricted master problem has a feasible solution by construction.

8.2. Subproblem

Once we have solved the restricted master problem optimally, we would like to know if there are columns we have not considered yet, but which may reduce the optimum objective function value of the restricted master problem when added. In other words, we have to look for decision variables z_{hs}, which correspond to schedules that have not been considered so far, with a negative reduced cost. The reduced cost is defined to be

$$rc(\mu, \pi, \lambda) = -\mu_h + \sum_{j \in V^h} \sum_{m \in M_j} \left(c_{jm} + p_{jm} \sum_{(i,j) \in \overline{E}} \pi_{ij} \right) x_{jms} + \sum_{(j,m_j,i,m_i) \in \mathscr{C}_{\overline{E}}} \sum_{m=\underline{m}_j}^{m_j} \lambda_{jm_j im_i} x_{jms}$$
$$+ \sum_{(i,m_i,j,m_j) \in \mathscr{C}_{\overline{E}}} \sum_{m=\underline{m}_j}^{m_j} \lambda_{im_i jm_j} x_{jms} + \sum_{j \in V^h} \left(\sum_{(j,i) \in \overline{E}} \pi_{ij} - \sum_{(i,j) \in \overline{E}} \pi_{ij} \right) C_{js},$$

where μ_h are the dual variables associated with (13), π_{ij} are the dual variables associated with (14), and $\lambda_{im_i jm_j}$ dual variables associated with (15).

We use a multiple pricing strategy in our tests which means that in every iteration we add one column for every subnetwork h for which a variable z_{hs} exists that has a negative reduced cost. Therefore, up to $|H|$ columns may be added to the restricted master problem per iteration. The column s to be added for subnetwork h is computed by solving the following optimization problem (we omit the index s for the sake of simplicity).

$$\min \quad rc(\mu, \pi, \lambda) \tag{17}$$

$$\text{s.t.} \quad \sum_{m \in M_j} x_{jm} = 1, \quad j \in V^h, \tag{18}$$

$$C_j - C_i - \sum_{m \in M_j} p_{jm} x_{jm} \geqslant 0, \quad (i,j) \in E^h, \tag{19}$$

$$C_j - \sum_{m \in M_j} p_{jm} x_{jm} \geqslant \mathrm{ES}_j, \quad j \in V^h, \tag{20}$$

$$\sum_{m=\underline{m}_i}^{m_i} x_{im} + \sum_{m=\underline{m}_j}^{m_j} x_{jm} \leqslant 1, \quad (i, m_i, j, m_j) \in \mathscr{C}_{E^h}, \tag{21}$$

$$\mathrm{EC}_j \leqslant C_j \leqslant LC_j, \quad j \in V^h, \tag{22}$$

$$x_{jm} \in \{0,1\}, \quad j \in V^h, \ m \in M_j. \tag{23}$$

The objective (17) equals the reduced cost. Note that, if the optimum objective function value of this minimization problem is non-negative, we add no column for subnetwork h in the current iteration. Note also that, if no column is added for any of the subnetworks, the column generation procedure terminates. The constraints (18) make sure that a unique mode is assigned to each of the activities in the subnetwork h (compare (2)). Due to (19), the precedence constraints within a subnetwork are respected (compare (3)). Because of (20), none of the activities in the subnetwork can be started before its earliest start time. Note that this set of constraints is not necessarily redundant for the source and for those activities $j \in V^h$ for which one or more immediately preceding activities do belong to another subnetwork than subnetwork h. The set of constraints (21) equals the set of cuts for activities which belong to subnetwork h. Because these cuts tighten the bounds derived from the LP-relaxation, they may speed up the optimization when a branch-and-bound procedure is employed. Finally, (22) and (23) define the domains of the decision variables. Recall, once a column is found for adding it to the restricted master problem, its objective function coefficient is determined using (11).

The trick of our approach is that the subproblem (17)–(23) can be solved using standard software, if the subproblem is sufficiently small. The size of the subproblem measured in terms of the number of binary variables in the above model formulation is under our control by means of the parameter b that is used as input for decomposing the original network. Of course, this subproblem is a hard to solve problem in theory, but as long as fast computation times can be achieved in practice, this approach is justified. In some regard, this is aspect is similar to the well-known idea of Benders' decomposition where usually a hard to solve (master) problem must be solved, too, but computational results show that good results can be gained which justifies this approach.

9. Heuristic

Now we will show how to derive a heuristic from the column generation procedure. The basic idea is to construct a feasible solution for the DTCTP from the optimal solution of the restricted master problem. Thus, when the column generation procedure terminates, we have computed as many feasible solutions as we have solved restricted master problems. From this set of feasible solutions, we are interested in the one with the lowest objective function value.

So, let us see what we can do once the restricted master problem is solved, i.e. once we have feasible values for the variables z_{hs}. The information that we utilize is the weighted average completion time of each activity j which is defined to be

$$\tilde{C}_j = \sum_{s \in S^{h(j)}} C_{js} z_{h(j)s}.$$

It should be remarked that the heuristic we will present works for acyclic networks only while the lower bounding procedure could be applied to cyclic networks as well.

9.1. Construction phase

In a first step, a feasible solution is constructed. This is done by considering the activities in topological order, i.e. an activity j is considered only if all immediate predecessors have already been considered before. Let j be the activity under consideration.

We start activity j as early as possible, i.e. the start time S_j of activity j is

$$S_j = 0,$$

if $j = 0$, and

$$S_j = \max_{i \in \mathcal{P}_j} \{C_i\},$$

otherwise. Activity j is scheduled in the "longest" mode $m_j \in M_j$ such that

$$S_j + p_{jm_j} \leqslant \tilde{C}_j,$$

which yields

$$C_j = S_j + p_{jm_j}$$

to be the completion time of activity j. Note that such a mode m_j must exist, because using the solution obtained by solving the restricted master problem means that $\tilde{C}_j$ defines a feasible schedule using some weighted average processing times for the activities.

9.2. Improvement phase

The feasible solution just obtained may be improved further by assigning even "longer" modes to some of the activities. Let m_j be the current mode assigned to activity j. The improvement phase iteratively proceeds as follows:

Step 1: Compute the earliest start times ES_j and the latest completion times LC_j for each activity using the current modes M_j.

Step 2: Determine the set $\mathcal{J}$ of activities to which a "longer" mode can be assigned while keeping all other mode assignments as they are. Formally, this set is

$$\mathcal{J} = \{j \in V | m_j > \underline{m}_j \text{ and } \mathrm{LC}_j - \mathrm{ES}_j \geqslant p_{j(m_j-1)}\}.$$

Step 3: If the set $\mathcal{J}$ is empty then stop. A local optimum is found. Otherwise, choose one activity out of $\mathcal{J}$ for assigning a "longer" mode to it. If $\mathcal{J}$ contains more than a single element, we apply a simple priority rule that takes into account the ratio of the cost improvement to the increase of duration of an activity. More formally, we compute

$$\Delta_j = (c_{jm_j} - c_{j(m_j-1)})/(p_{j(m_j-1)} - p_{jm_j})$$

for each $j \in \mathcal{J}$ and choose

$$j^* = \arg\max\{\Delta_j | j \in \mathcal{J}\},$$

where remaining ties are broken favoring the activity with the lowest number. Then, a new mode

$$m_{j^*} = m_{j^*} - 1$$

is assigned to activity j^* and we return to Step 1.

10. Computational studies

We carried out extensive computational studies in order to evaluate the performance of the network decomposition-based lower and upper bounds. The algorithms were coded in GNU C programming language and run on a Pentium II with 300 MHz processor, 64 MB RAM and Linux operating system. In addition to that, the (mixed-integer) linear programming solver CPLEX 5.0 was employed.

10.1. Test-bed 1: Large instances

To test the presented methods, no standard test-bed is available from the literature that could be claimed to be a hard set of instances for testing heuristics and algorithms for computing lower and upper bounds. Hence, we created a test-bed by our own (which is available from the authors upon request).

For generating activity networks, we generated AoA networks as described by Akkan et al. [3] and converted them into AoN networks in a straightforward manner. This is done, because the complexity index CI is defined for AoA networks, but not for AoN networks, and CI is known to have a strong impact on the hardness to solve a DTCTP instance. Recall that a brief description of CI is given in the literature review (see Section 3).

It is remarkable to note that the network generator presented in [3] allows to use the complexity index CI as input which has been an open problem so far. The AoA networks that are used have 17 nodes. We have generated networks with a complexity index CI $\in \{13, 14\}$ and with a coefficient of network complexity CNC $\in \{5, 6, 7, 8\}$ which is defined to be the ratio of the number of arcs to the number of nodes, see [19].

Depending on the complexity index CI and the coefficient of network complexity CNC, the result of the network generation procedure gives instances of a particular size measured by the number of activities. Table 1 shows which instance sizes were actually used in our study.

Three different types of cost functions were used. We studied concave (ccv), convex (cvx), and hybrid (hyb) ones where hybrid means that the cost function is neither concave nor convex.

The number of modes per activity may also affect computational performance. Hence, we studied two intervals from which the number of modes per activity is randomly chosen with uniform distribution. The intervals are $|M_j| \in \{[2;10],[11;20]\}$.

The details of how the modes of an activity are generated are as follows: First the number of modes, m, is generated, let us say from DU(2, 10) (i.e., discrete uniform distribution with parameters 2 and 10). Then, the durations of these modes are randomly generated between 3 and 123 as follows: The range 3–123 is divided into intervals of size 4 and a simple randomized rule is used to decide whether one of the modes will have a duration within that interval. If so, the duration is generated within the interval using Discrete Uniform distribution. After all the durations are determined, the costs of the modes are generated sequentially, starting with that of the minimum-cost mode, c_m, which comes from DU(5, 15). Given the duration, cost pair (d_k, c_k), for mode k and d_{k-1}, c_{k-1} is calculated as $\lfloor c_k + s_k(d_k - d_{k-1}) \rfloor$, where s_k is the randomly generated slope. For convex cost functions, s_{k-i} is generated from $U(s_k, s_k + S)$, where S is the maximum change in slope per mode and set to be equal to 3. s_{m-1} (the minimum slope) is set to be 1. For the concave functions, s_{m-1} is randomly generated as $1 + u(m-1)S$, where $u \sim U(0.75, 1.25)$ (so that the initial slope is large enough to allow for smaller slopes for the other modes), and then S_{k-1} is generated from $U(\max(1, (s_k - S)), s_k)$. Then, c_{k-1} is calculated as $\lfloor c_k + s_k(d_k - d_{k-1}) \rfloor$, For the hybrid functions, we randomly determine the number of times the slope will increase/decrease compared to the previous one. If the number of modes, m, is larger than 3, then the number of slope increases, s^+, is generated from DU$(1, (m-3))$, and the number of slope decreases, s^-, equals $m - 2 - s^+$. If m equals 3 then with equal

Table 1
Minimum/maximum number (n) of activities (test-bed 1)

	CI = 13	CI = 14
CNC = 5	85/85	85/85
CNC = 6	102/102	102/102
CNC = 7	111/119	115/119
CNC = 8	128/135	128/136

probability, we have $s^+ = 1$, $s^- = 0$ or $s^+ = 0$, $s^- = 1$. The rest of the mechanism is similar to those of convex and concave functions described above.

Finally, the deadline T of the project was assumed to be of interest. The deadline was calculated as follows. We computed the minimum possible project duration $T_{\min}$ of an instance when using the "shortest" modes. Likewise, we computed the minimum possible project duration $T_{\max}$ of an instance when using "longest" modes. From this, we determined $T = T_{\min} + \theta(T_{\max} - T_{\min})$, where $\theta \in \{0.15, 0.30, 0.45, 0.60\}$ was chosen.

For each parameter level combination, we generated 10 random instances which gives a total of $2 \times 4 \times 3 \times 2 \times 4 \times 10 = 1920$ large instances.

All tests were done by using a value $b = 60$ for decomposing the networks.

10.2. Test-bed 2: Wide range of CI values

For optimum solution procedures for the DTCTP, the complexity index is considered to have a strong impact on the performance. Thus, it is interesting to see, if this holds for the column generation procedure as well.

Similar to how the first test-bed has been created, we defined a second test-bed by generating AoA networks with 17 nodes and CNC = 2 using the network generator developed by Akkan et al. [3]. Since we want to study the complexity index, we used a wide range of CI values, i.e. we used CI $\in \{0, 4, 5, 6, 9, 10, 11, 14\}$. Note that AoA networks with 17 nodes cannot have a larger complexity index than 14 (see, for example, [3]). This gives small networks with a number of activities as provided in Table 2.

Again, we used three different types of cost functions (ccv, cvx, and hyb), two intervals for choosing the number of modes per activity ($|M_j| \in \{[2;10], [11;20]\}$), and four deadline settings $\theta \in \{0.15, 0.30, 0.45, 0.60\}$. For each parameter level combination, 10 random instances were generated which resulted in a total of $8 \times 3 \times 2 \times 4 \times 10 = 1920$ small instances.

As before, $b = 60$ was used to decompose the networks.

10.3. Computational results

Several aspects are worth to have a look at

- *The number of subnetworks.* To get a feeling into how many subnetworks a given network is decomposed, we provide this information.
- *The number of iterations.* Column generation is an iterative procedure that solves the master problem over and over again until the stopping criterion is met. The number of times this happens is provided.
- *The total number of columns.* Upon termination, the master problem consists of a certain number of columns. Since the number of possible columns is exponentially large, it is of interest how many columns are actually involved.
- *The number of removed modes.* The preprocessing procedure possibly eliminates some modes and it is of interest how effective this elimination procedure really is. Hence, we study the percentage of removed modes measured as

Table 2
Minimum/maximum number (n) of activities (test-bed 2)

CI								
0	4	5	6	7	9	10	11	14
29/30	34/35	34/36	35/37	36/36	37/39	36/42	37/42	38/42

$$100 \times \frac{\sum_{j \in V} d_j}{\sum_{j \in V} (|M_j| - 1)},$$

where d_j is the number of modes of activity j deleted during preprocessing and M_j is the set of modes before preprocessing.

- *The number of cuts.* Additional cuts help to tighten the lower bound and it may be an interesting piece of information how many cuts exist (no matter if they occur in the master problem or in the subproblems).
- *The run-time performance.* One of the most interesting performance indicators is the run-time performance measured in CPU-seconds. All figures given include preprocessing time.
- *The solution gap.* The most important performance measure is the solution gap that remains between the upper and the lower bound. This deviation is measured in terms of percentages as given by

$$100 \times \frac{\mathrm{UB} - \mathrm{LB}}{\mathrm{UB}},$$

where UB is the objective function value of the heuristic solution and LB is the lower bound obtained.

- *The improvement of simple approaches.* Another way to computer lower bounds is to use the idea of network decomposition much more straightforwardly. What we refer to as simple decomposition uses the network decomposition as presented to create subnetworks. Then a discrete time–cost tradeoff problem is solved optimally for each of these subnetworks. Formally, this means solving an instance of the model (17)–(23) optimally for each subnetwork h where all parameters μ_h, π_{ij}, and $\lambda_{im_i jm_j}$ being zero. The sum of optimum objective function values over all subnetworks $h \in H$ defines a lower bound denoted as LB_0. Since these subproblems are DTCTPs, optimum solution procedures from the literature can be applied. Hence, the subnetworks may be larger than those that are created with $b = 60$. We used $b = 300$ to compute LB_0 and compared this lower bound with the lower bound LB computed with column generation using $b = 60$. Of course, preprocessing is applied as well. The ratio

$$\frac{\mathrm{LB}}{\mathrm{LB}_0}$$

will be provided and can be interpreted as the improvement of column generation over simple decomposition.

- *Comparison with other heuristics.* The feasible solutions computed with our approach may be compared with the results of other heuristics. To the best of our knowledge, the best heuristics for the DTCTP are the iterated local search heuristics developed by Akkan [2]. He presents two algorithms, a so-called "kick" heuristic and a so-called "restart" heuristic. We applied both algorithms. For comparing the obtained results, we provide average figures of the ratio

$$\frac{\mathrm{UB}'}{\mathrm{UB}},$$

where UB' is the upper bound computed with one of the iterated local search procedures and UB is the column generation based upper bound. Thus, a ratio greater than one indicates that the local search result has been improved. In addition to that, we will give, for each local search procedure, the percentage of all instances for which the column generation based heuristic gives an upper bound less than the upper bound of the local search algorithm.

Tables 3–8 show the impact of the problem parameters CI, CNC, θ, $|M_j|$, and the type of cost function, respectively, on the computational results. Our main focus is on the large instance (test-bed 1), because these instances definitely have a size for which heuristics and lower bounds are needed, because optimum solution procedures fail to solve them.

Table 3
Computational results depending on CI (test-bed 1)

	13	14
Avg. # subnets	22.94	22.65
Avg. # iterations	28.61	27.52
Avg. # columns	302.92	293.44
Avg. % removed modes	23.18	22.73
Avg. # cuts	12.61	0.64
Avg. run-time	5.57	5.29
Avg. solution gap	7.44	6.21
Avg. LB_0 improvement	6.01	7.40
Avg. kick improvement	1.08	1.00
% kick improvements	48.96	53.85
Avg. restart improvement	0.98	0.97
% restart improvements	13.65	16.87

Table 4
Computational results depending on CI (test-bed 2)

	0	4	5	6	9	10	11	14
Avg. # subnets	12.25	15.02	15.33	15.62	16.18	16.22	16.27	16.47
Avg. # iterations	12.68	15.91	16.11	16.45	16.54	16.77	17.53	15.42
Avg. # columns	90.31	123.96	130.13	131.85	139.09	142.58	144.29	133.56
Avg. % removed modes	3.34	1.60	1.49	1.43	17.36	14.98	14.61	11.90
Avg. # cuts	0.59	3.75	3.60	1.82	144.40	154.66	121.60	0.61
Avg. run-time	0.46	0.72	0.75	0.78	1.06	1.09	1.06	0.76
Avg. solution gap	2.41	4.50	4.51	4.50	6.02	6.77	6.72	3.82
Avg. LB_0 improvement	43.74	36.69	34.02	37.16	18.32	19.25	18.67	35.39
Avg. kick improvement	1.01	1.01	1.01	1.01	1.00	1.00	1.00	1.01
% kick improvements	73.33	69.58	63.75	67.08	56.25	48.33	52.50	70.42
Avg. restart improvement	1.00	1.00	1.00	1.00	0.99	0.99	0.99	1.00
% restart improvements	49.17	40.42	38.33	39.58	31.25	30.00	27.50	43.75

Table 5
Computational results depending on CNC (test-bed 1)

	5	6	7	8
Avg. # subnets	19.18	21.85	23.61	26.53
Avg. # iterations	26.70	28.21	28.51	28.84
Avg. # columns	231.55	283.99	322.41	354.76
Avg. % removed modes	25.31	21.91	22.24	22.36
Avg. # cuts	7.27	14.16	4.01	1.06
Avg. run-time	2.98	4.67	6.24	7.81
Avg. solution gap	5.59	6.77	7.38	7.54
Avg. LB_0 improvement	6.92	6.64	6.44	6.81
Avg. kick improvement	1.01	1.00	1.17	1.00
% kick improvements	60.00	46.88	50.00	48.75
Avg. restart improvement	0.99	0.98	0.95	0.98
% restart improvements	21.46	13.54	14.58	11.46

Regarding the complexity index, CI, we observe the following major issues (see Tables 3 and 4). The run-time is slightly affected if CI values are varied. The results for the small instances with a wide range of CI

Table 6
Computational results depending on θ (test-bed 1)

	0.15	0.30	0.45	0.60
Avg. # subnets	22.79	22.79	22.79	22.79
Avg. # iterations	33.11	28.75	26.03	24.37
Avg. # columns	348.80	306.36	279.43	258.11
Avg. % removed modes	7.42	15.43	26.95	42.01
Avg. # cuts	26.50	0.00	0.00	0.00
Avg. run-time	7.45	5.62	4.68	3.96
Avg. solution gap	8.47	7.54	6.42	4.85
Avg. LB_0 improvement	2.01	7.36	10.35	7.09
Avg. kick improvement	1.04	1.04	1.04	1.04
% kick improvements	36.25	52.50	53.54	63.33
Avg. restart improvement	0.98	0.97	0.97	0.99
% restart improvements	14.17	14.17	13.54	19.17

Table 7
Computational results depending on $|M_j|$ (test-bed 1)

	[2:10]	[11;20]
Avg. # subnets	12.22	33.37
Avg. # iterations	32.98	23.15
Avg. # columns	230.13	366.22
Avg. % removed modes	27.24	18.67
Avg. # cuts	3.75	9.50
Avg. run-time	5.45	5.40
Avg. solution gap	8.30	5.35
Avg. LB_0 Improvement	2.75	10.65
Avg. kick improvement	1.07	1.02
% kick improvements	54.69	48.12
Avg. restart improvement	0.98	0.97
% restart improvements	14.90	15.62

Table 8
Computational results depending on cost function type (test-bed 1)

	ccv	cvx	hyb
Avg. # subnets	22.85	22.88	22.64
Avg. # iterations	23.34	31.91	28.94
Avg. # columns	233.36	369.23	291.93
Avg. % removed modes	22.58	23.21	23.07
Avg. # cuts	6.61	6.42	6.85
Avg. run-time	4.40	6.52	5.36
Avg. solution gap	9.62	4.60	6.24
Avg. LB_0 Improvement	8.33	4.08	7.70
Avg. kick improvement	0.97	1.09	1.07
% kick improvements	45.78	50.31	58.12
Avg. restart improvement	0.98	0.98	0.98
% restart improvements	15.78	9.38	20.62

values, however, indicates that a larger CI value need not result in larger run-time, because CI = 14 gives faster computation times than CI $\in \{6, 9, 10, 11\}$. The solution gap usually increases, if CI gets larger, but

again we see that CI = 14 gives better results than all other CI values except for CI = 0. These results are remarkable, because up to now an increasing complexity index was assumed to make instances harder to solve and requiring more computational effort. Note, simple decomposition is clearly outperformed for all CI values, but it gives poor results compared to column generation especially when CI is low or when it is high. Thus, the proposed approach seems to fit well especially for hard instances. Even for large instances, the solution gap is low (an overall average below 7%) while the run-time effort is small (below 6 CPU-seconds). On average, the objective function values of our heuristic are slightly better than the "kick" heuristic results, and our results are slightly worse than the "restart" heuristic solutions. Compared to the "kick" heuristic, the column generation based heuristic gives a better result for about 50% of all large instances in the test-bed (test-bed 1) and the figures are much better in favor of the column generation procedure for small instances (test-bed 2). The findings do not indicate that any of the three heuristics that are tested outperforms the others significantly. Nevertheless, two points should be stressed: First, our approach not only gives a heuristic solution, but a lower bound as well. And second, we have observed that the "restart" heuristic requires several minutes of computation time for large instances (the "kick" heuristic takes several seconds) while the column generation procedure terminates after a few seconds. This justifies the claim that, compared to other heuristics, the column generation based heuristic is competitive.

The coefficient of network complexity (CNC), which in fact is a measure for the number of activities, shows the following main effects (see Table 5). As one would expect, the number of subnetworks and the number of columns grows when more activities are involved. It is interesting, however, that the number of iterations is more or less robust. Also, the preprocessing procedure eliminates about the same percentage of modes no matter what size the instance has. Surprisingly, the number of cuts does not grow when instances get larger in terms of the number of activities. In fact, only a few cuts exist for large instances. The run-time as well as the solution gap increases (slightly) when larger instances are solved. Nevertheless, the results are still very satisfying. CNC has almost no effect on the improvement of column generation over simple decomposition. So, we can conclude that even if instances have a large number of activities (see Table 1) the run-time as well as the solution gap is acceptable and the proposed approach is indeed capable of solving large instances.

With regard to the time horizon θ, we have the following important findings (see Table 6). Increasing the size of the instance (measured in terms of time periods involved) significantly reduced the number of iterations, the number of columns and the required run-time effort. The number of modes that can be eliminated by means of preprocessing grows significantly when θ and thus T grows. It should be noted that the proposed cuts exist only for small time horizons. Most importantly, the solution gap decreases when θ increases. Thus, we can conclude that our procedure is capable to solve instances no matter what time horizon is defined.

The number of modes per activity also affects the results (see Table 7). As one would expect, the number of subnetworks and the number of columns increases when $|M_j|$ increases. But, the number of iterations decreases. Overall, the run-time is not affected. Instances are considered to be harder to solve the more modes exist. Our procedure, however, seems to perform especially good when many modes exist, because we can observe that the solution gap decreases and that the simple decomposition is outperformed much more clearly in such cases. So, our procedure appears to be well suited to hard instances again where hardness is measured in terms of the number of modes this time.

Focusing on the cost function type (see Table 8), we can see that hard instances are those which have a concave cost function, because the solution gap is highest for such instances (but still below 10% on average). In contrast to this, convex cost functions make instances easy to solve near optimally.

Finally, one should note that we have tried to apply the Lagrangean relaxation approach proposed by Akkan [1] as well, but it turned out that it is not able to solve instances of the size we have used. So, the presented approach is indeed the only available method so far that yields sophisticated lower and upper bounds for large instances in short computation time. An emphasis is on short here, because although we

generated all network paths (during preprocessing) and we solved a sequence of mixed-integer linear programs, which might seem to cause long computation time in theory, it turned out that in practice our approach is indeed efficient. Given that our approach is fast while giving almost identical heuristic objective function values as state-of-the-art heuristics (which do not yield lower bounds), the proposed column generation procedure makes a contribution with regard to both, lower and upper bounds for the DTCTP.

11. Conclusions

The discrete time–cost tradeoff problem (DTCTP) is a well-established project scheduling problem. It has attracted several researchers in the past decades. As a consequence, important issues such as the complexity status are well researched. For solving the problem optimally, several procedures have been proposed, but none of these is able to solve large and hard instances measured in terms of, say, the complexity index or the number of activities. So, heuristics are clearly needed for attacking such instances. For evaluating heuristics, lower bounds are crucial. Surprisingly, only little work has been published on heuristics and, to the best of our knowledge, there has been just one single contribution on sophisticated lower bounds which is based on Lagrangean relaxation. This Lagrangean relaxation approach gives both, a heuristic solution and a lower bound.

In this paper, the proposed approach also computes a feasible solution and, at the same time, a lower bound for evaluating the solution. Thus, the contribution of this paper is stronger than just a heuristic or just a lower bounding procedure. Network decomposition is investigated to attack large instances in a divide and conquer manner. Then, column generation techniques are applied to compute lower bounds while, within each iteration of the column generation procedure, a feasible solution is constructed.

A computational study examines the performance of the proposed method in depth. Several problem parameters are studied which are assumed to have a strong impact on the hardness of a problem instance: the complexity index, the coefficient of network complexity, the project deadline, the number of modes per activity, and the cost function type. It turned out that instances can now be solved which are far beyond what has previously been attacked (networks with up to 138 activities, a complexity index of up to 14, and up to 20 modes per activity). The overall average solution gap is less than 7% obtained in less than 6 CPU-seconds on a Pentium II PC with 300 MHz. The heuristic results are competitive with state-of-the-art heuristics (which do not give a lower bound). This and the fact that the Lagrangean relaxation procedure which gives lower and upper bounds is not able even to deal with instances of the tested size proves that the proposed procedure makes a significant contribution.

Acknowledgements

The authors are indebted to anonymous referees for their comments and suggestions which improved the readability of the paper. The research of the second author has been supported by the Deutsche Forschungsgemeinschaft (Grant Dr. 170/6).

References

[1] C. Akkan, A Lagrangian heuristic for the discrete time–cost tradeoff problem for activity-on-arc project networks, Working Paper, Koç University, Istanbul, 1998.

[2] C. Akkan, Iterated local search algorithms for the discrete time–cost tradeoff problem, Working Paper, Koç University, Istanbul, 1999.

[3] C. Akkan, A. Drexl, A. Kimms, Generating an acyclic directed graph with a given complexity index by constraint logic programming, Journal of Logic and Algebraic Programming, to appear.

[4] W.W. Bein, J. Kamburowski, M.F.M. Stallmann, Optimal reduction of two-terminal directed acyclic graphs, SIAM Journal on Computing 21 (1992) 1112–1129.
[5] N. Billstein, F.J. Radermacher, Time–cost optimization, Methods of Operations Research 27 (1977) 274–294.
[6] P. De, E.J. Dunne, J.B. Ghosh, C.E. Wells, The discrete time–cost tradeoff problem revisited, European Journal of Operational Research 81 (1995) 225–238.
[7] P. De, E.J. Dunne, J.B. Ghosh, C.E. Wells, Complexity of the discrete time–cost tradeoff problem for project networks, Operations Research 45 (1997) 302–306.
[8] V.G. Deineko, G.J. Woeginger, Hardness of approximation of the discrete time–cost tradeoff problem, Operations Research Letters 29 (2001) 207–210.
[9] E. Demeulemeester, W. Herroelen, S.E. Elmaghraby, Optimal procedures for the discrete time/cost trade-off problem in project networks, European Journal of Operational Research 88 (1996) 50–68.
[10] S.E. Elmaghraby, Resource allocation via dynamic programming in activity networks, European Journal of Operational Research 88 (1992) 50–86.
[11] H. Frank, I.T. Frisch, R. van Slyke, W.S. Chou, Optimal design of centralized computer networks, Networks 1 (1970) 43–58.
[12] D.R. Fulkerson, A network flow computation for project cost curves, Management Science 7 (1961) 167–178.
[13] R.T. Harvey, J.H. Patterson, An implicit enumeration algorithm for the time/cost tradeoff problem in project network analysis, Foundations of Control Engineering 4 (1979) 107–117.
[14] T.J. Hindelang, J.F. Muth, A dynamic programming algorithm for decision CPM networks, Operations Research 27 (1979) 225–241.
[15] J.E. Kelley, Critical path planning and scheduling: Mathematical basis, Operations Research 9 (1961) 296–320.
[16] J.E. Kelley, M.R. Walker, Critical Path Planning and Scheduling: An Introduction, Mauchly Associates, Ambler, PA, 1959.
[17] M. Krishnamoorthy, N. Deo, Complexity of the minimum-dummy-activities problem in a PERT network, Networks 9 (1979) 189–194.
[18] R.H. Möhring, F.J. Radermacher, Substitution decomposition for discrete structures and connections with combinatorial optimization, Annals of Discrete Mathematics 19 (1984) 257–356.
[19] T.L. Pascoe, Allocation of resources—CPM, Revue Française de Recherche Opérationelle 38 (1966) 31–38.
[20] D.R. Robinson, A dynamic programming solution to the cost-time tradeoff for CPM, Management Science 22 (1975) 158–166.
[21] B. Rothfarb, H. Frank, D.M. Rosenbaum, K. Steiglitz, D.J. Kleitman, Optimal design of offshore natural-gas pipeline systems, Operations Research 18 (1970) 992–1020.
[22] M. Skutella, Approximation algorithms for the discrete time–cost tradeoff problem, Mathematics of Operations Research 23 (1998) 195–203.
[23] A. Sprecher, A. Drexl, Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm, European Journal of Operational Research 107 (1998) 431–450.
[24] M.M. Syslo, On the computational complexity of the minimum-dummy-activities problem in a PERT network, Networks 14 (1984) 37–45.