A reduction tree approach for the Discrete Time/Cost Trade-Off Problem

Rob Van Eynde[a], Mario Vanhoucke[a,b,c,*]

[a] *Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, Ghent 9000, Belgium*
[b] *Vlerick Business School, Reep 1, Ghent 9000, Belgium*
[c] *UCL School of Management, University College London, 1 Canada Square, London E14 5AA, United Kingdom*

Abstract

Cite this paper as: "Van Eynde, R., and Vanhoucke, M. (2022). A reduction tree approach for the discrete time/cost trade-off problem. Computers and Operations Research, 143, 105750. (doi: 10.1016/j.cor.2022.105750)" The Discrete Time/Cost Trade-Off Problem is a well studied problem in the project scheduling literature. Each activity has multiple execution modes, a solution is obtained by selecting a mode for each activity. In this manuscript we propose an exact algorithm to obtain the complete curve of non-dominated time/cost alternatives for the project. Our algorithm is based on the network reduction approach in which the project is reduced to a singular activity. We develop the reduction tree, a new datastructure that tracks the modular decomposition structure of an instance at each iteration of the reduction sequence. We show how it is related to the complexity graph of the instance. Several exact and heuristic algorithms to construct a good reduction tree are proposed. Our computational experiments show that the use of the reduction tree provides significant speedups when compared to the existing reduction plan approach. Although the new approach does not outperform the best performing branch-and-bound procedure from the literature, the experiments show that incorporating modular decomposition can provide significant performance improvements for solution algorithms, showing potential for developing improved hybridised procedures to solve this challenging problem type.

Keywords:
Discrete time/cost trade-off, Reduction tree, Modular decomposition

*Corresponding author
Email addresses: rob.vaneynde@ugent.be (Rob Van Eynde), mario.vanhoucke@ugent.be (Mario Vanhoucke)

1. Introduction

Time/cost trade-off problems have been widely studied in project scheduling literature. In this type of problems, activities have multiple execution options or modes and the cost of executing an activity is inversely related to its duration. Several *continuous* versions of this problem have been investigated, among other linear, convex and concave relations between cost and duration. For an overview of these studies, we refer to the literature review of Vanhoucke and Debels (2007).

In the *discrete* time/cost trade-off problem (DTCTP), the problem of the current study, each activity has a finite set of execution modes. Each mode has a duration and a cost and modes with a shorter duration have a higher cost. There exist three variants of this problem. In the *budget problem*, the objective is to minimize the project duration while keeping the total cost below a specified budget limit. The *deadline problem* considers the case where the total project cost needs to be minimized while ensuring that the project finishes within a deadline. In the *curve problem*, the objective is to construct the complete trade-off curve with all efficient time/cost alternatives for the project.

The contribution of this research is threefold. First, we establish the relevance of our research by summarizing the literature on the DTCTP and its extensions, and add a discussion on the importance of the problem along five criteria. Second, we propose a new exact solution approach for solving the DTCTP by introducing the reduction tree into a modular decomposition approach. Finally, we validate our approach with various computational experiments. More specifically, we implement four exact and one heuristic algorithms to construct a reduction tree, and compared them with and existing reduction plan approach and a very efficient branch-and-bound procedure from the literature.

In this paper we propose a new exact solution procedure that incorporates modular decomposition into an existing algorithm from literature. The paper is structured as follows. In Section 2 we outline the contribution of our research and provide an overview of the state of the literature regarding the DTCTP. Section 3 introduces the necessary notation and definitions. Section 4 discusses the reduction tree, which is the core building block of our approach. Section 5 describes the basic construction algorithm for reduction trees and the solution algorithm that computes the trade-off curve using a reduction tree. In Section 6 we propose different algorithms to construct a good reduction tree. In Section 7, our approaches are evaluated and benchmarked against the algorithms of Demeulemeester et al. (1996) and Demeulemeester et al. (1998).

2. Relevance

This section gives a detailed description of the relevance of our research and its importance in academia and practice by giving a full overview of the problem (and its extensions) and highlighting its importance in a discussion from five different angles.

2.1. Literature review

The three variants of the DTCTP are strongly NP-hard (De et al., 1997). In the previous century, many researchers have developed exact solution approaches for the problem. A literature review and a detailed discussion of the problem and solution strategies before 1995 is given in De et al. (1995). The authors discuss among other dynamic programming, network decomposition and network reduction approaches. The ideas from their discussion on modular decomposition will be used in this paper.

Table 1 provides an overview of literature on the DTCTP that focuses on the period from 1995 to now. The "Objective" column describes the objective function of the studied problems and consists of 5 subcolumns. The Curve (C), Deadline (D) and Budget (B) objectives were already discussed in the introduction. It is clear that most of the listed studies address the curve or the deadline problem. For the Net present value (N) objective, the mode costs are considered as cash outflows and cash inflows are modeled as payments. The objective is to optimize the net present value of these cash flows, either from the perspective of the project owner, the project contractor or both. Due to its relevance in practice, this objective has been studied by quite some researchers. Among the Other objectives (O), Hazır et al. (2010b) and Li et al. (2020) address the schedule robustness in the face of uncertainty. A second alternative objective is to minimize the maximal gap between cash inflows and outflows (He et al., 2017; Ning et al., 2017). Last, Aouam and Vanhoucke (2019) address the objective of contract design that maximizes the owner value, given different effort levels of the contractor.

The "Problem type" column also consists of 5 subcolumns. The first column represents Time/Cost Trade-off problems (TCT), all studies in this review consider this problem. In some studies, this base problem is extended with additional problem types. In the Payment Scheduling problem (PS), one takes into account cash outflows (activity costs) and cash inflows (progress payments upon completion of milestones or of the complete project). The review shows that payment scheduling is a relevant extension, as it has been studied by many researchers. A second extension is Time-Switch constraints (TS) in which activities can only start in a specified time interval in a cycle. For instance, these constraints can be used to model a work week where activities cannot start on weekend days. A third extension is Work Continuity constraints (WC), in which the total cost is the combination of the activity costs and work continuity costs. These occur for subsets of activities and grow based on the length of the time-interval between the first and last activity of that set. The last column considers Other (O) extensions. Deineko and Woeginger (2001) and Szmerekovsky and Venkateshan (2012) consider irregular costs, which allow to model a variety of costs and cash flows, and can for instance be used for payment scheduling. Chassiakos and Sakellaropoulos (2005) extend the problem with generalized precedence relations, external time constraints and earliness bonuses and tardiness penalties. Tardiness penalties are also included by Hazır et al. (2010b) and Said and Haouari (2015). Aminbakhsh and Sonmez (2016) extend the problem with indirect costs that are not related to the activity costs.

Last, Aouam and Vanhoucke (2019) combine the time/cost trade-off problem with performance contract design.

The fourth column considers the problem under "Uncertainty". Most of the research do not include uncertainty and assume that all the project information is deterministic. The subcolumn Duration (D) represents the studies in which the mode durations can deviate from their expected durations. The subcolumn Cost (C) shows the studies in which the activity cost is seen as uncertain. The Effort (E) subcolumn considers the extension where a contractor can exert additional effort to keep the project cost and makespan on track. The uncertainty lies in how effective this effort is in reducing these performance indicators. Although some research exists on this uncertainty, the attention remains limited in literature.

The "Methodology" column quantifies which approach is used to solve the problem. The first and second subcolumns represent Exact (E) and Heuristic (H) solution algorithms. The subcolumn Other (O) show other methodologies. These include results in approximability (Skutella, 1998; Deineko and Woeginger, 2001), lower bounds (Akkan et al., 2005), computational complexity (Grigoriev and Woeginger, 2004) and simulation (Hazır et al., 2010b). The exact approaches consist of, among others, network reduction approaches (Demeulemeester et al., 1996), branch-and-bound algorithms (Demeulemeester et al., 1998; Değirmenci and Azizoğlu, 2013), Benders decomposition (Hazır et al., 2010a) and linear programming relaxation (Hafızoğlu and Azizoğlu, 2010).

The column "Applications" lists which research includes applications to real life cases or instances. In this review, six studies translate the research to practical applications such as construction and installation cleaning. These studies show that the research on the basic DTCTP and its extensions such as time/switch and work continuity constraints is relevant for practitioners, as will be discussed more in the next section.

The last column "Dataset" lists the number of instances on which the researchers tested their solution methods. Note that the references labelled with "–" are studies on computational complexity and approximability, which contain no experimental results using test instances.

We will now review the exact solution procedures that were published after 1995. Demeulemeester et al. (1996) propose an exact solution procedure that can be used for the curve, deadline and budget problem. The core idea is to reduce the project network to a single arc (i.e. activity) by applying a sequence of reduction operations that are specified by a reduction plan. The modes of this final activity correspond to the complete trade-off curve of the project. Their idea was in turn inspired by the research of Bein et al. (1992), who developed the theory of network reductions and introduce the three types of reduction operations that are required to reduce a network to a single arc. Our research builds further on this network reduction approach by generalizing the concept of a reduction plan to modular decomposition. A different solution approach was introduced by Demeulemeester et al. (1998). It is a branch-and-bound procedure that uses a horizon-varying approach in which the total curve is obtained by iteratively minimizing the cost given different deadlines. To the best of our

knowledge, no new exact solution procedures have been developed for the basic curve problem since this research.

Nevertheless, the table shows that in recent years, new research has been done on exact procedures for other variants of the DTCTP. Vanhoucke et al. (2002) and Vanhoucke (2005) propose branch-and-bound algorithms for the extension with time-switch constraints. A horizon-varying exact approach for the work continuity extension was introduced by Vanhoucke (2006). Chassiakos and Sakellaropoulos (2005) extend the base problem with generalized precedence constraints, external time constraints and a bonus/penalty structure and solve the problem with linear/integer programming. For the deadline problem, Akkan et al. (2005) provide lower and upper bounds using column generation techniques, while Hafızoğlu and Azizoğlu (2010) introduce a branch-and-bound algorithm. For the budget problem, Hazır et al. (2010a) propose a Benders decomposition approach and Değirmenci and Azizoğlu (2013) discuss a branch-and-bound procedure. Hazır et al. (2011) extend the Benders decomposition approach from Hazır et al. (2010a) to incorporate uncertainty, with the objective to construct robust schedules. Szmerekovsky and Venkateshan (2012) compare four integer programming models for the DTCTP with irregular costs. Aouam and Vanhoucke (2019) solve the problem in a multi-staged solution approach, in which the contractor subproblem is solved with the exact algorithm of Demeulemeester et al. (1998).

2.2. Discussion

This section elaborates on the literature review presented in the previous section to show the relevance of the discrete time/cost trade-off problem in general, and the results of the current research study in particular for both academia and practice, as explained along the five following paragraphs.

Fundamental problem: First and foremost, the DTCTP is one of the two fundamental problems in the literature on project scheduling with resources. The project scheduling literature has split the use of project resources into two basic classes. First, the presence of limited *renewable* resources has resulted in a wide stream of research to solve the well-known resource-constrained project scheduling problem (RCPSP) for which many research summaries are available. The DTCTP of the current study makes use of the second basic resource type, known as *non-renewable* resources, which has been investigated much earlier than the RCPSP, but is up to today still a challenging problem with potential to numerous extensions. Both resource types are still highly relevant for further research, as they can lead to additional insights and new solution procedures that combine the two types of resources to e.g. *doubly-constrained resources* or *partially renewable resources*.

Modular decomposition: The idea of modularly decomposing the DTCTP is not new and has already been suggested in the summary article of De et al. (1995). However, the best of our knowledge, this manuscript is the first study that investigates the value of incorporating it into an optimal solution algorithm. Our approach combines modular decomposition with the network reduction approach, which leads to the idea of the *reduction tree* approach used in

Authors	Objective					Problem type					Uncertainty			Methodology			Applications	Dataset
	C	D	B	N	O	TCT	PS	TS	WC	O	D	C	E	E	H	O		
Elmaghraby (1993)			x			x								x	x			1
Erenguc et al. (1993)				x		x								x				140
Demeulemeester et al. (1996)	x	x	x			x								x				16
Feng et al. (1997)	x					x									x		Construction	1
Demeulemeester et al. (1998)	x					x								x				1,800
Skutella (1998)	x	x	x			x										x		-
Deineko and Woeginger (2001)	x					x										x		-
Vanhoucke et al. (2002)		x				x		x						x	x		Installation cleaning	12,000
Grigoriev and Woeginger (2004)		x				x	x			x						x		-
Akkan et al. (2005)		x				x									x	x		3,840
Chassiakos and Sakellaropoulos (2005)		x				x				x				x	x		Highway construction	1
Vanhoucke (2005)		x				x		x						x			Installation cleaning	30,000
Vanhoucke (2006)	x					x			x					x			Tunnel construction	2,763
Vanhoucke and Debels (2007)	x			x		x		x	x						x			108,000
He and Xu (2008)		x		x		x	x								x			1
HE et al. (2009)				x		x	x								x			3,240
He et al. (2009)		x		x		x	x								x			4,320
Hafızoğlu and Azizoğlu (2010)		x				x								x	x			480
Hazır et al. (2010a)			x			x								x				240
Hazır et al. (2010b)			x		x	x				x	x					x		36
Hazır et al. (2011)		x				x						x		x	x			1,920
He et al. (2012)				x		x	x								x			48,600
Sonmez and Bettemir (2012)	x					x									x			10
Szmerekovsky and Venkateshan (2012)	x			x		x	x			x				x				720
Değirmenci and Azizoğlu (2013)			x			x								x	x			360
He et al. (2014)				x		x	x								x			19,440
Said and Haouari (2015)		x				x				x	x				x			108
Aminbakhsh and Sonmez (2016)	x					x				x					x			87
He et al. (2017)		x			x	x	x								x			3,240
Ning et al. (2017)		x			x	x	x				x				x			87,480
Agdas et al. (2018)	x					x									x		Construction	8
Li et al. (2018)	x					x									x			600
Aouam and Vanhoucke (2019)					x	x				x			x	x				15
Leyman et al. (2019)		x		x		x	x								x			157,464
Li et al. (2020)	x				x	x					x				x			2,880

Table 1: Summary literature

our study. The experiments of Section 7 show that the approach leads to significant improvements compared to the reduction plan approach without modular composition

Hybrid algorithms: Of course, our new algorithm does not always outperform all algorithms available in the literature. For example, the computational experiments of Section 7.3 shows our approach still cannot compete with a dedicated branch-and-bound (BNB) algorithm from the literature that relies on a modified version of the elegant algorithm proposed by Fulkerson (1961). Despite this, we believe the modular decomposition holds promise since it indicates that applying modular decomposition into this BNB search will likely provide performance improvements. Therefore, modular decomposition is considered as a general approach rather than a algorithm-specific solution procedure, and our results show that it can improve the running time of an algorithm by applying it to the node reduction approach. We believe this opens interesting avenues to future research areas to hybridise the modular decomposition in other exact algorithms and heuristics to solve the DTCTP.

Phase transitions: The research of this study can also provide additional insights into the drivers of the complexity of the DTCTP. For the RCPSP, researchers have tested their algorithms on well-designed benchmark instances for decades, and report the performance of new algorithms taking problem-specific indicators into account. Herroelen and De Reyck (1999) has presented the concept of *phase transitions* and state that indicators that measure the network topology and the resource scarceness show a clear *complexity pattern* which can be used to predict the time an algorithm needs to solve the problem. Many research studies on the RCPSP have investigated the impact of network and resource characteristics on the solution quality and problem complexity (see. e.g. (Kolisch et al., 1995; De Reyck and Herroelen, 1996; Vanhoucke et al., 2008; Van Eynde and Vanhoucke, 2022). In a recent study of Coelho and Vanhoucke (2018) on the RCPSP, the authors expressed their hope that future researchers will take project-specific characteristics much better into account than they did before, which clearly indicates the importance of understanding the real drivers of problem complexity. For the DTCTP, similar, albeit much less research has been performed that show that the *complexity index* (CI) is the main indicator to predict the complexity of this problem (Kamburowski et al., 1992; De et al., 1995; Demeulemeester et al., 1996; De Reyck and Herroelen, 1996). Our research study advances these insights by showing that when the reduction tree is used, its depth can replace the CI as explanatory variable for the CPU-time, which clearly contributes to a better understanding of the complexity of a DTCTP instance.

Practical relevance: The previous paragraphs clearly illustrate that the DTCTP has not only a rich research history but also holds a promising future for further research. However, apart from its relevance in academia, we also believe that the problem is highly relevant in business environments. Developing advanced solution procedures are of course not always directly applicable to practice, but we are nevertheless convinced that fundamental research is necessary to create efficient and simplified procedures for the DTCTP and

its extensions that may occur in practice. In the literature review, we have already highlighted some interesting DTCTP extensions such as the presence of week/weekend calendars (time/switch constraints) Vanhoucke and Debels (2007), the incorporation of cash flow optimisation using the net present value objective (He et al., 2009, 2017; Leyman et al., 2019), an agency perspective view on the problem for contracting (Aouam and Vanhoucke, 2019) or even the inclusion of problem stochasticity (Hazır et al., 2010b; Said and Haouari, 2015; Ning et al., 2017). However, the practical relevance of the DTCTP can also be illustrated by the fact that the problem has been incorporated into two well-known business games. A first game is knows as CPSim (Piper, 2005), which is adapted from an exercise called CAPERTSIM that was developed by North American Aviation, Inc., Autonetics Education and Training Department. CAPERTSIM was available from the IBM SHARE library, and was adapted by the Harvard Business School in 1974 and renamed PLANETS II (planning and network simulation). A second well-known game is PSG (Project Scheduling Game) (Vanhoucke et al., 2005) that has been used since its introduction in 2005 in various academic and commercial project management trainings at business schools in Belgium, the UK and China. The PSG can be played with any realistic project network tailored to the needs and wishes of the game players, and can easily be extended to another project by simply changing the input file of the network under study. This makes it possible to tune the training to the participants' needs and to make the game very recognizable to the participant. This game has been used in two classroom experiments consisting of master and MBA students who played the game during a project management training. First, Wauters and Vanhoucke (2016) have investigated the impact of project complexity and uncertainty on the solution strategies of the participants. More specifically, the authors investigated the impact of the participants' perception on complexity/uncertainty on how time/cost trade-offs in projects are managed. They pay special attention to the influence of the actual and perceived complexity and uncertainty and the cost repercussions when reality and perception do not coincide. More recently, Servranckx and Vanhoucke (2021) have searched for the essential skills in a data-driven project management classroom experiment in which the PSG has played a central role. They identified seven project management skills and statistically investigated the link between these skills and performance of the student on five case studies, among which the PSG was one of them. The results indicated that time/cost planning is key for good risk analysis and project control, and they showed that a combination of soft and hard skills is key in solving the different case studies.

We believe that this section clearly shows the importance of fundamental research on a challenging problem like the DTCTP. In the next section, the problem and its detailed features related to graph theory is outlined in detail, which will then be used in Section 4 to present our new solution approach.

Instance	
N	Set of activities, indexed by i
μ_i	Mode set of activity i
$m_{ik}(d_{ik}, c_{ik})$	Mode k of activity i, with duration and cost
P_i, S_i	Direct predecessors and successors of i
P_i^*, S_i^*	Direct and indirect predecessors and successors of i

Graph	
V, A	Set of vertices and arcs
$a_i(v, w)$	The arc corresponding to activity i, from vertex v to w
A_X	Set of arcs corresponding to the set of activities X
$I(v), O(v)$	Set of incoming and outgoing arcs of vertex v
$I^*(v), O^*(v)$	Set of predecessor and successor arcs of vertex v

Table 2: Symbols

3. Definitions

An instance consists of a set of activities N, each activity i has a set of execution modes μ_i. Each mode $m_{ik} \in \mu_i$ has a corresponding duration d_{ik} and cost c_{ik}. Without loss of generality, we assume that all modes are non-dominated i.e. all modes have a different duration and $d_{ik} < d_{il}$ implies that $c_{ik} > c_{il}$. Furthermore, precedence relations between activities exist. We denote the direct predecessors and successors of activity i by P_i and S_i respectively. An activity i can only be started when all predecessors in P_i are finished. The set of both direct and indirect predecessors and successors are denoted by P_i^* and S_i^* respectively.

3.1. Graph related definitions

There exist two formats to represent the precedence relations of the instance. In the activity-on-the-node (AON) format, the instance is represented by an acyclic digraph where each vertex corresponds to an activity and the arcs represent the precedence relations. In this manuscript we will use the activity-on-the-arc (AOA) format, as the node reduction approach of Bein et al. (1992) is developed for this format. An instance in the AON format can be converted into the AOA format, but this may require the introduction of dummy arcs. The problem of finding the AOA representation with the minimum number of arcs is NP-hard (Michael and Kamburowski, 1993). Fortunately, it is possible to construct an AOA representation in polynomial time that does not increase the complexity of an instance (Kamburowski et al., 1992; De Reyck and Herroelen, 1996).

In the AOA format, an instance is represented by a two-terminal acyclic digraph $D(V, A)$ where each activity in N is represented by an arc in A. As the graph has one source vertex s and one sink vertex t, we will also use the notation s,t-digraph. Given a (sub)set of activities X, A_X denotes the set of arcs that correspond to these activities. If arc $a_i = (u, v)$ precedes arc $a_j = (v, w)$, its

corresponding activity i is a predecessor of activity j. $I(v)$ and $O(v)$ denote the set of incoming and outgoing arcs of vertex v respectively. $I^*(v)$ denotes the set of all arcs that lie on a path from s to v and $O^*(v)$ denotes the set of all arcs that lie on a path from v to t. We will also refer to $I^*(v)$ and $O^*(v)$ as the set of predecessor and successors arcs respectively. We assume that the graph is topologically ordered, i.e. for any arc $(v, w) \in A$ it follows that $v < w$. Given a subset of vertices V', $D[V']$ is the subgraph of D induced by V'. For a subset of arcs A', $D[A']$ is the subgraph consisting of the vertices that are incident to an arc in A' and the arc set A'.

A set of activities M is a **module** if for any two activities $i, j \in M$, $P_i^* \setminus M = P_j^* \setminus M$ and $S_i^* \setminus M = S_j^* \setminus M$. In words, all activities in a module have the same indirect predecessors and successors outside that module. In the example network of Figure 1 the sets $\{1, 2\}$ and $\{6, 7\}$ are modules. An immediate consequence of this definition is that for a module M and an activity $b \notin M$, either $M \subseteq P^*(b)$, $M \subseteq S^*(b)$ or $M \cap S^*(b) = M \cap P^*(b) = \emptyset$. M is a **strong module** if for any other module M', either $M \cap M' = \emptyset$ or $M \subseteq M'$ or $M' \subseteq M$. In Figure 1, $\{1, 2\}$ and $\{6, 7\}$ are both strong modules.

In the s, t-digraph D, a vertex v **dominates** a vertex w if every path from s to w includes v. If in addition $v \neq w$, v **properly dominates** w. Dually, a vertex w **reverse-dominates** a vertex v if every path from v to t includes w. Furthermore, if $v \neq w$, w **properly reverse-dominates** v. In the example Figure 1, vertex b properly dominates c and c properly reverse-dominates b.

In this manuscript we wil use the **complexity graph**, an auxiliary structure that is constructed from the two-terminal dag.

Definition 1 (Complexity graph, (Bein et al., 1992))**.** *The complexity graph $C(D)$ of a two-terminal dag D is defined by $(v, w) \in C(D)$ if and only if there exists a path $P(v, w)$ in D such that for every $u \in P(v, w)$; u neither properly dominates w nor properly reverse-dominates v.*

In the figure below, an s, t-digraph D and its complexity graph of $C(D)$ are shown. (a, b) is an arc in $C(D)$ as there exists an (s, b)-path that does not include a and an (a, t)-path that does not include b.

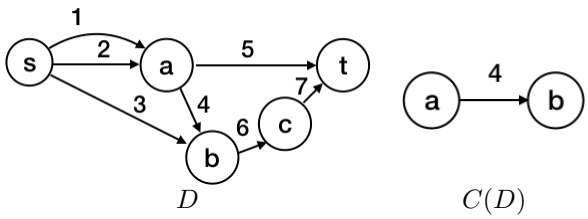


Figure 1: An instance D and complexity graph $C(D)$

3.2. Reduction operations

The solution approach in this paper relies on applying a sequence of reduction operations until one activity remains, corresponding to the arc (s, t)

in D. This approach was introduced by Demeulemeester et al. (1996), in this subsection we recall the definitions that are relevant here.

To completely reduce an instance, three types of reductions are required: parallel, serial and node reductions. Each of the reductions are illustrated in Figure 2. Two activities i, j are **parallel** if their corresponding arcs have the same start and end vertices v and w. A **parallel reduction** $p(i, j)$ replaces these activities by a new activity l with the mode set μ_l consisting of a all non-dominated modes in the set $\{(\max(d_{ik}, d_{jl}), c_{ik} + c_{jl}) : \forall m_{ik} \in \mu_i, m_{jl} \in \mu_j\}$. In the graph, the arcs a_i and a_j are replaced by a new arc $a_l = (v, w)$. In the illustration, D_1 shows the result of applying the parallel reduction $p(1, 2)$ on D from Figure 1.

Two activities i, j are **in series** if $a_i = (u, v), a_j = (v, w)$ and the node v has no other incoming or outgoing arcs. A **series reduction** $s(i, j)$ replaces i and j by a new activity l with mode set μ_l consisting of all non-dominated modes in the set $\{(d_{ik} + d_{jl}, c_{ik} + c_{jl}) : \forall m_{ik} \in \mu_i, m_{jl} \in \mu_j\}$. In D, arcs a_i and a_j are replaced by a new arc $a_l = (u, w)$ and vertex v is deleted. In Figure 2, applying the series reduction $s(6, 7)$ on D_1 results in D_2.

The last operation is the **node reduction**. A forward node reduction can be applied on an activity i if its arc $a_i = (v, w)$ is the only outgoing arc of node v and v has multiple incoming arcs. A backward node reduction can be applied on an activity i if $a_i = (v, w)$ is the only incoming arc of node w and w has multiple outgoing arcs. Given an activity i with $a_i = (v, w)$, a mode $m_{ik} \in \mu_i$ and a boolean variable f (indicating forward or backward), the node reduction operator $n(i, m_{ik}, f)$ can be defined. If $f =$ True, a_i is removed and each activity j with an arc $a_j = (u, v) \in I(v)$ is replaced by a new activity j' with arc $a_{j'} = (u, w)$ and with mode set $\mu_{j'}$. If a_j is the first arc in $I(v)$, the mode set of its new activity j' is defined by $\{m_{j'l} = (d_{jl} + d_{ik}, c_{jl} + c_{ik}) : \forall m_{jl} \in \mu_j\}$. For any other activity h with arc $a_h \in I(v)$, the mode set of the new activity h' is defined by $\{m_{h'l} = (d_{hl} + d_{ik}, c_{hl}) : \forall m_{hl} \in \mu_h\}$. The reduction operation with $f =$ False also removes a_i, but replaces all activities in $O(v)$ by new activities in a similar fashion as the forward operation. We will sometimes use the notation $n(i, f)$ when the choice of mode is unimportant and $n(i)$ when only the activity i is important. In the example below, D_3^a and D_3^b respectively show the result of the node reductions $n(8, \text{False})$ and $n(9, \text{True})$ on D_2.

If a (sub)graph can be reduced to a single arc using only series and parallel reductions, this (sub)graph is **sp-reducible**. If at least one node reduction is required, it is **irreducible**. D_1 and D_2 are irreducible, while D_3^a and D_3^b are sp-reducible. Bein et al. (1992) established that to reduce an s, t-digraph D to a single arc, for each arc in its complexity graph $C(D)$, a node reduction needs to be applied to one of its incident vertices. Therefore, the minimum number of node reductions for D equals the size of the minimum vertex cover in $C(D)$. It follows that a graph is only sp-reducible if $C(D)$ is empty. De Reyck and Herroelen (1996) introduced the term complexity index (CI) to denote this minimum number of node reductions.

The solution approach of Demeulemeester et al. (1996) is built upon the concept of a **reduction plan**. This is a sequence of reduction operations such

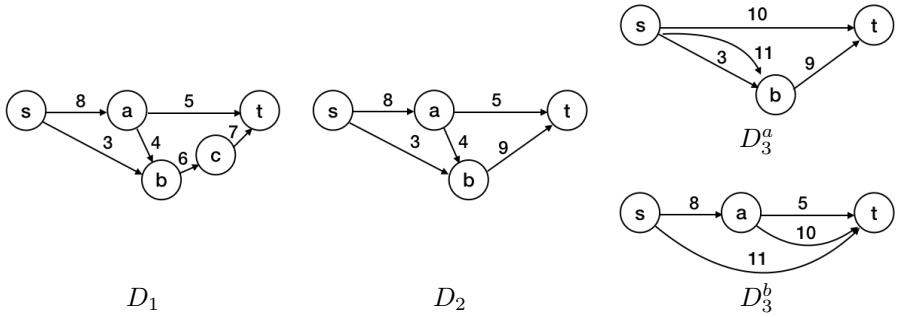


Figure 2: Illustration of reduction operations

that applying all reductions in the order of the plan results in a completely reduced instance (i.e. a single activity). The performance of this approach strongly depends on the number of node reductions in this plan, which cannot be smaller than the CI of the instance.

4. The reduction tree

The outline of the next two sections is summarised in Figure 3. In this section we will discuss the reduction tree, which constitues the core structure for our optimal procedure that is discussed in Section 5. In Subsection 4.1 we discuss some theoretical insights that will allow us to efficiently construct the reduction tree using the complexity graph of the instance. In Subsection 4.2 we provide the formal definition of the reduction tree and illustrate it with an example.

4.1. Modules in complexity graph

First we show that modules are related to the triconnected components in a digraph. A **connected component** of a digraph is an inclusion-wise maximal subgraph for which there exists an undirected path between any two of its vertices. The graph $C(D)$ in Figure 4 contains two connected components, with vertex sets $\{a, b\}$ and $\{c, d\}$ respectively. The definition of a triconnected component in a digraph is more elaborate.

Definition 2 (Triconnected component)**.** *A subgraph $D'(V', A')$ is a triconnected component in the s,t-digraph $D(V, A)$ if it satisfies the following conditions:*

1. *V' contains a node pair (v, w) such that v dominates all nodes in $V' \setminus w$ and w reverse-dominates all nodes in $V' \setminus v$*
2. *If $x \in V' \setminus \{v, w\}$ and there exists a path from v to x or from x to w that contains y, then $y \in V'$*
3. *A' is nonempty and each arc in A' has both endpoints in V'*
4. *For each vertex $x \in V' \setminus \{v, w\}$: $I(x) \subseteq A'$ and $O(x) \subseteq A'$*

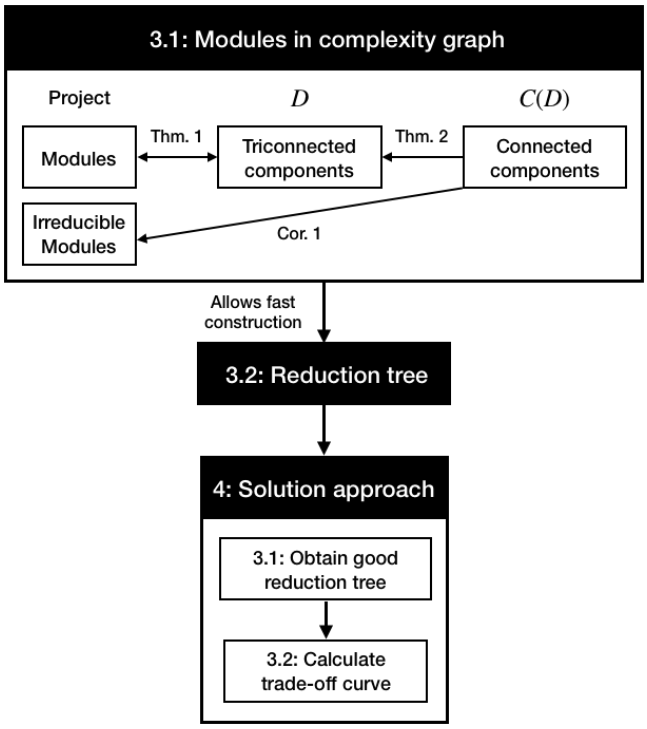


Figure 3: Overview structure Sections 4 and 5

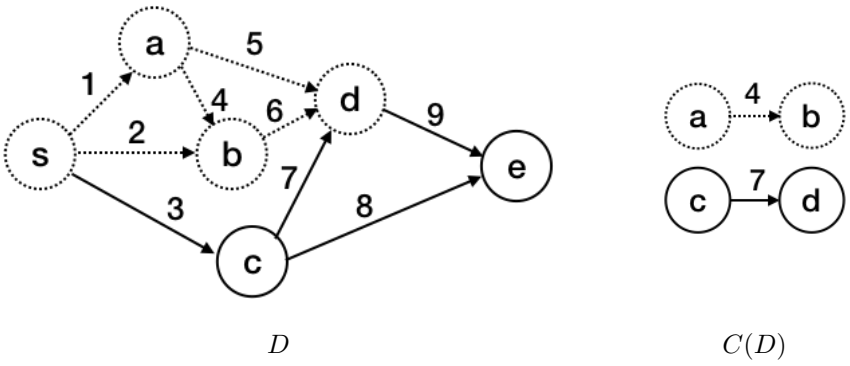


Figure 4: An AOA graph and its complexity graph

Figure 4 shows an example s,t-digraph D with its corresponding complexity graph $C(D)$. The subgraph $T(V', A')$ that is indicated by the dotted lines is a triconnected component, with $V' = \{s, a, b, d\}$ and $A' = \{1, 2, 4, 5, 6\}$. We will illustrate the four conditions on this example. For Condition 1, one can verify that (s, d) is the node pair such that s dominates all nodes in $V' \setminus d$ and d reverse-dominates all nodes in $V' \setminus s$. To illustrate Condition 2, assume that $a \in V'$. As there is a path from a to d that includes b, it follows that $b \in V'$. Condition 3 is easily verified as all arcs in A' have both endpoints in V'. The last condition is also satisfied, as all incoming and outgoing arcs of a and b are in A'.

The complete s,t-digraph is always a triconnected component. The reader can verify that in the example, the complete graph D satisfies the four conditions. Given a subgraph $S \subseteq D$, we will denote the smallest triconnected component in D that contains S by $D \triangledown S$. Recall that in the example above, T is the triconnected component corresponding to the dotted subgraph. For the subgraph S_1 with vertices a, b and arc 4, the reader can verify that $T = D \triangledown S_1$. Furthermore, for the subgraph S_2 with vertices c, d and arc 7: $D = D \triangledown S_2$.

Theorem 1. *In the AOA format, a set of activities M is a nonempty module if and only if the subgraph $D[A_M] = D'(V', A')$ is a triconnected component.*

Proof. ONLY IF: We start by proving that M is a module only if all conditions of a triconnected component are satisfied.

Condition 1: Suppose that M is a module but that there does not exist one node v that dominates all nodes in V'. This means that there exist at least two nodes v_1 and v_2 such that $I(v_1) \cap A' = \emptyset$, $I(v_2) \cap A' = \emptyset$ and both nodes have at least one outgoing arc (a_i and a_j respectively) in A'. As $I(v_1) \neq I(v_2)$, activities i and j don't have the same predecessors and thus M cannot be a module. It follows from this contradiction that there has to exist a node v that dominates all nodes in V'. The proof for w and reverse-domination follows the same structure.

Condition 2: Suppose that M is a module and there exists a node $x \in V'$ for which there is a (x, w)-path P with a node $y \in P$ that is not in V'. It follows

that none of the arcs of $I(y)$ and $O(y)$ are in A'. As an (x, y)-path exists and M is a module, all activities in M need to have the activities corresponding to $O(y)$ as successors. However, there exist activities with corresponding arcs in A' that are incident to vertices on the path from y to w, which cannot have the activities corresponding to $O(y)$ as successors. This is a contradiction, confirming the second condition. The proof for the other direction has the same structure.

Condition 3: This follows from the definition of $D[A_M]$ and the fact that M is nonempty.

Condition 4: For an $x \in V' \setminus \{v, w\}$, it follows from the definition of $D[A_M]$ that at least one arc in $I(x)$ or $O(x)$ should be in A'. We will show that if there exists an $a_i \in I(x)$ that is in A', then $O(x) \subseteq A'$ and $I(x) \subseteq A'$. For the sake of brevity, we omit the proof for the case where there exists an $a \in O(x)$ that is in A', but it follows the same structure. Suppose that $O(x) \cap A' = \emptyset$, and take an arc $a_j = (x, y) \in O(x)$. It follows that there is at least one arc in A' that is incident to y, call it a_l. It follows that even though a_i and a_l are in A', $i \in P^*(j)$ and $l \notin P^*(j)$, which would imply that M is not a module. Now suppose that a strict subset $S \subset O(x)$ is in A'. It follows that the activities corresponding to $O(x) \setminus S$ are successors of activity i, but not of the activities corresponding to S. This again contradicts the fact that M is a module, leading to the conclusion that $O(x) \subseteq A'$. Now suppose that there exists an arc $a_j \in I(v)$ that is not in A'. As $a_i \in A'$ and $O(v) \subseteq A'$, this implies that j is a predecessor of all activities corresponding to $O(v)$, but not of activity i, implying that M is not a module. It follows that $I(v) \subseteq A'$.

IF: We now prove that M is a nonempty module if $D[A_M]$ is a triconnected component.

Suppose that we have a triconnected component $D'(V', A')$, where M is the set of activities corresponding to the arc set A'. One can easily verify for any activity $i \in M$ that $P^*(i) \setminus M$ is the activity set corresponding to $I^*(v)$ and $S^*(i) \setminus M$ is the activity set corresponding to $O^*(w)$. Therefore, the set of activities corresponding to T is a module. □

Theorem 1 is illustrated in Figure 4, where the activities corresponding to the arc set A' form a module. We will now show that triconnected components in D are related to connected components in $C(D)$. Suppose we have a triconnected component T in D. Due to the first property of the triconnected components, it follows that $T \cap C(D)$ will not be connected to $(D \setminus T) \cap C(D)$. This is illustrated in the right hand side of Figure 4, which shows the complexity graph $C(D)$. Again, T corresponds to the triconnected component indicated by the dotted lines. $T \cap C(D)$ and $(D \setminus T) \cap C(D)$ are indicated by dotted and complete lines respectively and these parts are not connected. The following theorem formalises this relation between the triconnected components of D to the connected components in $C(D)$.

Theorem 2. *Each connected component in $C(D)$ corresponds to a unique triconnected component in D.*

Proof. Take two distinct connected components C_1 and C_2 in the complexity graph $C(D)$ and $T_1 = D\triangledown C_1$, $T_2 = D\triangledown C_2$. We only need to prove that $T_1 \neq T_2$. Suppose that T_1 and T_2 are equal (call it T), with terminal vertices (v, w). As T is the minimal triconnected component containing C_1 and C_2, there exist no vertices in $T \setminus \{v, w\}$ that (reverse-)dominate C_1 or C_2. Therefore, the only way that C_1 and C_2 can be disconnected is because there are no paths in D between the components. However, this would imply that T could be split up in two smaller triconnected components, containing C_1 and C_2 respectively. This contradicts the fact that T is minimal, therefore confirming the theorem. □

Corollary 1. *Each connected component in $C(D)$ corresponds to a unique irreducible module.*

This corollary follows from the previous Theorem 2, the fact that each triconnected component corresponds to a module and that $C(D) \cap T$ is nonempty only if T corresponds to a irreducible module.

Figure 4 can be used as illustration. The instance contains two irreducible modules, $M = \{1, 2, 4, 5, 6\}$ and $M' = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, each corresponding to a triconnected component in D. On the right is the complexity graph $C(D)$, consisting of the connected components C induced by vertex set $\{a, b\}$ and C' induced by $\{c, d\}$. The reader can verify that C corresponds to M and C' to M'.

4.2. Reduction tree

In this subsection, we elaborate on a new datastructure, the reduction tree. This is a generalisation of the reduction plan, which allows efficient tracking of the modular decomposition structure at each stage in the reduction sequence.

Symbol	Explanation
R	Reduction tree
r	Vertex of R
$o(r)$	Reduction operation corresponding to r
$p(r)$	Parent of r
$c_r(x)$	Child at position x of vertex r
$c_r^{-1}(x)$	Position of vertex x in the child list of r
$\delta(r)$	Position of r in the depth-first search order of R
$\delta^{-1}(x)$	Tree vertex at position x in the depth-first search order of R

Table 3: Notation reduction tree

Before we provide the formal definition, some additional terminology is introduced. R denotes a reduction tree, we use r to refer to one of its nodes. Each $r \in R$ has a corresponding reduction operation, indicated by $o(r)$. r is an n-vertex if $o(r)$ is a node reduction. Let $p(r)$ be the parent of vertex r and $c_r(x)$ be the child on position x in the child list of parent r. $c_r^{-1}(x)$ denotes the position of vertex x in the child list of parent r. Furthermore, let $\delta(r)$ be the position of r in the depth first search (DFS) order of the tree, such that

$\delta(r_1) < \delta(r_2)$ if r_1 is visited before r_2 when traversing R with a depth first search. $\delta^{-1}(x)$ denotes the vertex at position x in the DFS order, $\delta^{-1}(0)$ refers to the root node. We use $D|R(r)$ to refer to the partially reduced instance that remains after applying the reductions in the sequence $o(\delta^{-1}(1)), \ldots, o(\delta^{-1}(r))$.

Definition 3 (Reduction tree)**.** *The reduction tree R is an ordered tree where each vertex $r \in R$ (except the root node) corresponds to a reduction operation on the instance. Furthermore, it has the following properties:*

1. *Only the root and n-vertices have children*
2. *The reduction sequence $o(\delta^{-1}(1)), o(\delta^{-1}(2)), \ldots, o(\delta^{-1}(|R|))$ is a reduction plan*
3. *Each n-vertex r corresponds to an irreducible strong module in $D|R(p(r))$, call this module M_r*
4. *For any vertex r, the reduction operation $o(r)$ is applied on activities in $M_{p(r)}$*
5. *Let M_i and M_j be two irreducible strong submodules of M_r in $D|R(r)$. If $M_i \subset M_j$, then $c_r^{-1}(i) < c_r^{-1}(j)$ and $M_l \subseteq M_j$ holds for all $l : c_r^{-1}(l) \in [c_r^{-1}(i), c_r^{-1}(j)]$*

Property 2 shows that the reduction tree is a generalisation of the reduction plan, as it can always be converted to a reduction plan by using the sequence $o(\delta^{-1}(1)), \ldots, o(\delta^{-1}(|R|))$.

We will now illustrate the reduction tree on the example of Figure 4. From its complexity graph, we can deduce that two node reductions need to be applied and that there are two irreducible strong modules. Figure 5 shows a reduction tree R for the instance. Each rectangle corresponds to one tree vertex r, its reduction operation $o(r)$ is shown in the upper right corner. Inside the rectangle is the graph that remains after applying that reduction operation, i.e. $D|R(r)$. According to the tree R, a backward node reduction $n(1, \text{False})$ will be applied on activity 1 and a forward node reduction $n(9, \text{True})$ will be applied on activity 9. Note that the indices of the vertices also show the DFS order of the tree.

R illustrates the 5 properties from Definition 3. Property 1 is satisfied as only the root (0) and n-vertices (1 and 5) have children. The graphs in the figure show that applying the reduction operations in the depth-first search order of R results in a completely reduced instance (i.e. $D|R(8)$). For Property 3, there are two n-vertices (indexed 1 and 5), their corresponding modules are $M_1 = \{1, 2, 4, 5, 6\}$ and $M_5 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ respectively. Property 4 is satisfied as the operations $o(2)$, $o(3)$ and $o(4)$ are applied on activities in M_1 and operations $o(6)$, $o(7)$ and $o(8)$ are applied on activities in M_5. Note that if a reduction operation is applied on some activities in the same module, the resulting activities are also part of that module. Property 5 is also satisfied as $M_1 \subset M_5$ and tree vertex 1 occurs before 5 in the child list of the root vertex.

The figure also illustrates that if an instance contains a module, it can first be completely reduced before considering the rest of the instance. $D|R(4)$ shows the partially reduced instance where M_1 is completely reduced to the single activity 14.

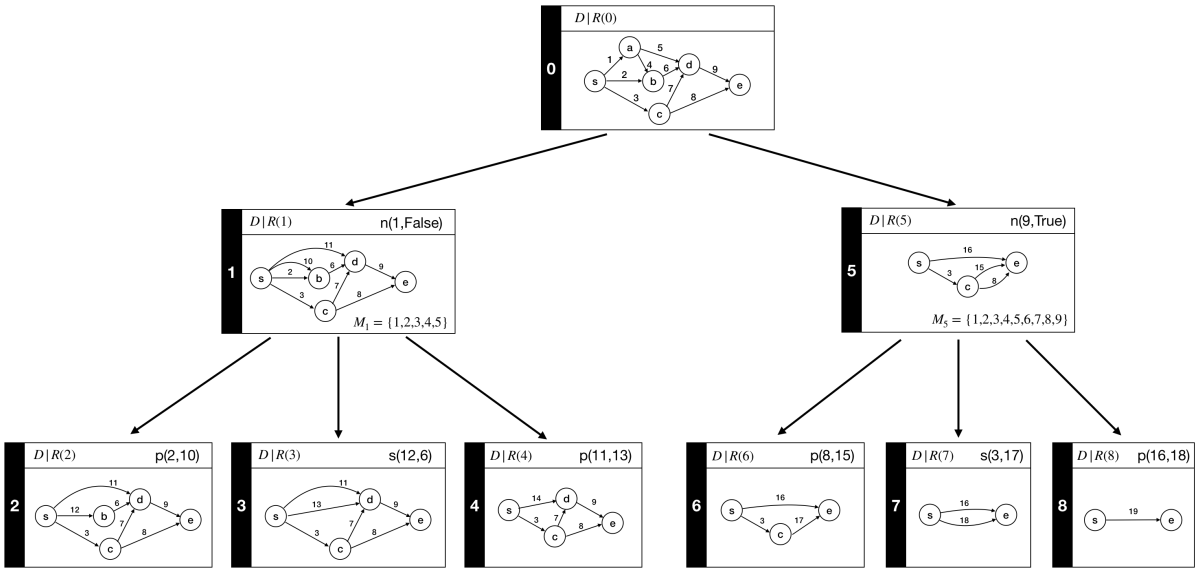


Figure 5: Reduction tree R with its reduction operations

5. Solution approach

Our approach is based on the concept that node reductions may partition the complexity graph in disconnected subgraphs, i.e. modules. First we describe how node reductions can partition an instance into modules and how this impacts the reduction tree. Based on this insight, we propose the core algorithm to construct a reduction tree. Second, we describe the solution algorithm that exploits the modular decomposition structure of the reduction tree to compute the complete trade-off curve.

5.1. Tree construction

Compared to the reduction plan approach, using the reduction tree may already lead to significant improvements for the computational effort as it tracks the modular decomposition structure of an instance. In this subsection we discuss how further improvements can be obtained by means of an additional observation: applying the node reduction operation may partition the graph in smaller modules.

We will show this by means of an example. Figure 6 shows the graph D of an instance on the left hand side, with its corresponding complexity graph $C(D)$ below. The instance $D|n(2, \text{False})$ on the right is the result of applying a backward node reduction on activity 2. In the original graph, only one irreducible strong module was present (i.e. the whole graph). After the node reduction, the remaining graph can be partitioned into two irreducible strong modules: $\{1, 5, 6, 12, 13, 16\}$ and $\{3, 4, 9, 10, 11, 14, 15, 17\}$. This is also reflected in the complexity graph. Applying the node reduction deletes node b in the

complexity graph, which results in the creation of two separate connected components.

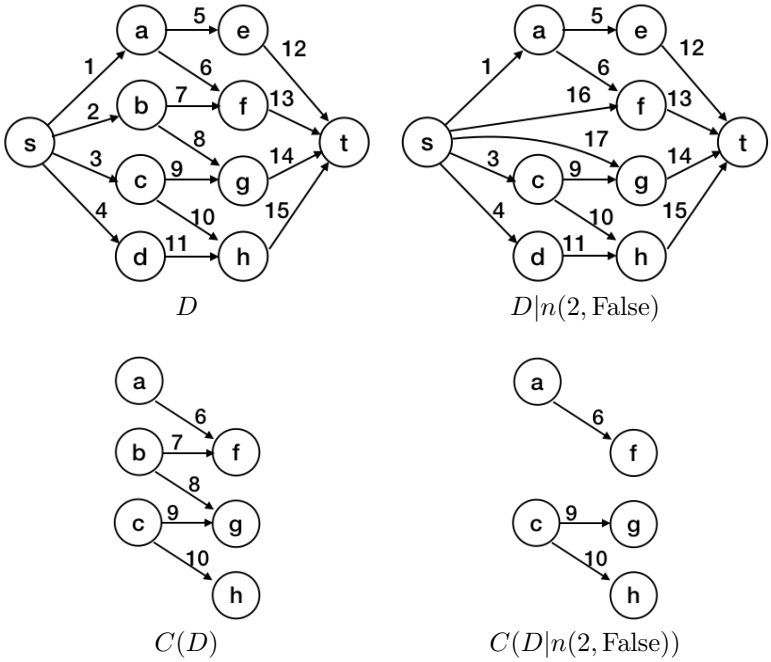


Figure 6: Example of modular partitioning

Algorithm 1 shows the base algorithm for the modular partitioning approach. It outputs a complete reduction tree for the instance. The reduction tree is initialised as a root node with a child node for each irreducible strong module in the initial graph. Each tree node that corresponds to an irreducible strong module is labeled as an L-node. A complete reduction tree can be obtained by applying Algorithm 1 on the root node. Recall that M_c denotes the module that corresponds to the tree node c.

At each child node c of the vertex r, the algorithm selects an activity to perform a node reduction (line 2) in the module M_c, after which it performs all possible series and parallel reductions (SP-reductions) in the module, call the remaining graph D'. If the remainder of the module (i.e. $M_c \cap D'$) consists of more than one activity, additional node reductions are required. Therefore, a recursive call is made on each of the irreducible strong submodules of $M_c \cap D'$ (line 10). Note that the order in which these submodules are considered respects Condition 5 from the Definition 3.

We illustrate the algorithm on the example from Figure 6. A complete reduction tree for the instance can be found in Figure 7. The top section of the figure shows the step-wise construction of this tree by the algorithm. In the tree, an L-node represents an irreducible strong module that is not completely reduced yet. As long as there are L-nodes in the tree, more node reductions are

Algorithm 1: dfsConstructTree(r)

```
for Each child c of r do
    Select an activity i ∈ M_c eligible for node reduction
    Apply node reduction n(i)
    o(c) ← n(i)
    for Each possible SP-reduction in M_c do
        Add new tree node c' as child of c
        Apply reduction operation and save it as o(c')
    D' ← remaining graph after reduction operations
    if |M_c ∩ D'| > 1 then
        for Each irreducible strong module M' ⊆ (M_c ∩ D') do
            Add new tree node c' as child of c, set M_c' = M'
            dfsConstructTree(c')
for Each possible SP-reduction in M_r do
    Add new tree node c' as child of r
    Apply reduction operation and save it as o(c')
```

required. For the sake of clarity, we added the modules corresponding to the L-nodes between brackets.

At the beginning, there is only one irreducible strong module. After the first node reduction (operation 1), the remaining instance can be partitioned into two irreducible strong modules. This is shown in the second tree where node 1 has two children. The algorithm makes a recursive call on the first child, where a node reduction and a sequence of SP reductions is applied, completely reducing the module. The third tree in the figure corresponds to this state. As the first module is reduced, the algorithm backtracks to vertex 1. From there, a recursive call is made on the second child of node 1, leading to a similar reduction sequence. After this sequence, the instance consists of two parallel activities, which are then reduced by operation 14. This complete reduction tree is the last tree in the figure. Note that we chose these specific node reductions to illustrate the effect of partitioning the instance in smaller modules. This complete reduction tree is the last tree in the figure. Note that the minimum number of node reductions is 3. However, thanks to the reduction tree structure, at most two node reductions need to be applied in sequence. The subproblems corresponding to the subtrees rooted at nodes 2 and 9 can be solved independently.

5.2. Trade-off curve calculation

Once a reduction tree is constructed, it is used as input for the algorithm that computes the complete trade-off curve of the project. It is similar to the algorithm described in Demeulemeester et al. (1996), but it is generalised such that it works for reduction trees. We start by describing this algorithm and then show how this can improve the computational efficiency in comparison

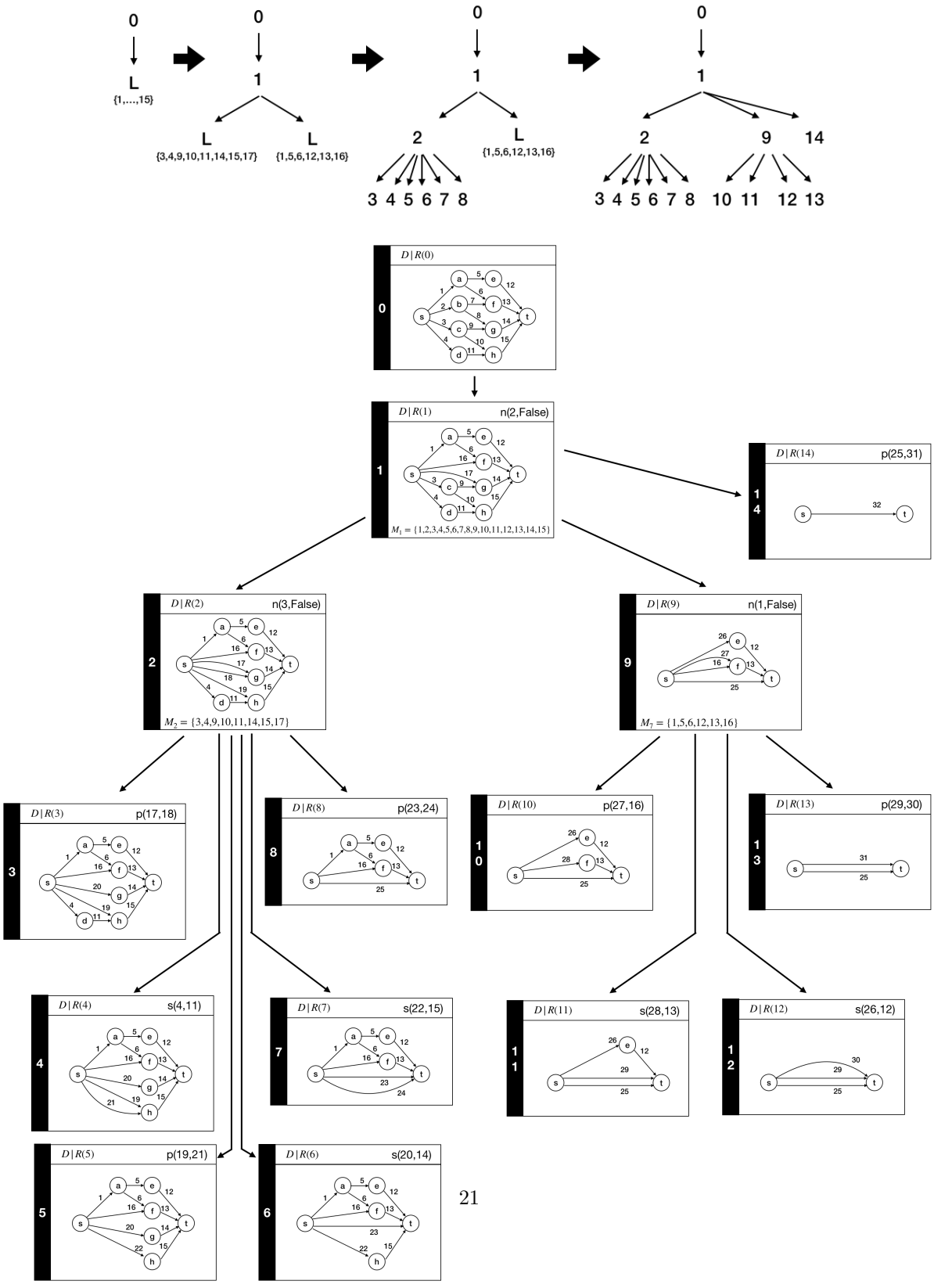


Figure 7: Reduction tree R with its reduction operations

to a reduction plan. An additional definition is required: lastAct(r) of a tree vertex r denotes the final activity that represents the complete module M_r. For the example in Figure 7, after the complete reduction of the module $M_2 = \{3, 4, 9, 10, 11, 14, 15, 17\}$ of tree node 2, this module is represented by the single activity 25, so lastAct(2)=25. The resulting graph is shown in tree node 8, the last child of tree node 2. Similarly, lastAct(9)=31 for tree node 9, as the completely reduced module M_9 is represented by activity 31.

Algorithm 2 exhibits the structure of the solution procedure. When it is called on a tree vertex r with corresponding module M_r, it will calculate the trade-off curve for this module. In other words, it completely reduces the module to the single activity $l = \text{lastAct}(r)$ such that the mode set μ_l corresponds to the trade-off curve of the module. The procedure iterates over all the children of r. If the operation $o(c)$ of the child is an SP-reduction, the algorithm applies this reduction (line 17). If the operation $o(c)$ is a node reduction on an activity i (line 3-15), the algorithm sets $l = \text{lastAct}(c)$ and applies the node reduction once for each mode in μ_i. After the application of the node reduction with a certain mode m_{ik}, the procedure makes a recursive call on this child node c. After this call μ_l contains the mode set of l given the node reduction $n(i, m_{ik})$. μ_l is then combined with the trade-off curve TC of the previous iterations. If m_{ik} was not the last mode in μ_i, all reduction operations in this subtree are reverted (lines 12-14), such that the sequence can be repeated for the next mode. The second function in Algorithm 2 describes this reversion procedure. Once all modes in μ_i have been considered, TC contains the complete trade-off curve for the module M_c. Therefore, TC is saved as the mode set μ_l of activity l.

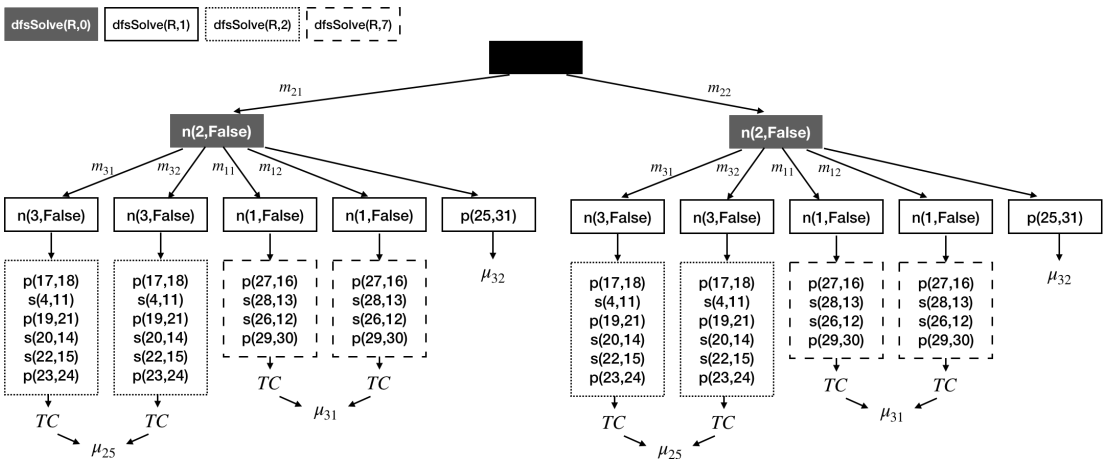


Figure 8: Example reduction tree solution algorithm

Figure 8 shows the working of Algorithm 2 when applied to the reduction tree of Figure 7. We assume that each activity has two modes. Each rectangle corresponds to a recursive call of the algorithm in which one or more reductions are applied. The top of the figure indicates to which recursive call the rectangles

Algorithm 2: Trade-off curve calculation

```
1  Function dfsSolve(R,r):
2      for c ∈ children of r do
3          if c is n-vertex then
4              TC ← ∅
5              i ← activity on which o(c) will be applied
6              f ← True (False) if o(c) is a forward (backward) node
                reduction
7              l ← lastAct(c)
8              for m_ik ∈ μ_i do
9                  Apply n(i, f, m_ik)
10                 dfsSolve(R,c)
11                 TC ← non-dominated modes in (TC ∪ μ_l)
12                 if k < |μ_i| then
13                     dfsRevert(R,c)
14                     Revert n(i, f, m_ik)
15             μ_l ← TC
16         else
17             Apply reduction o(c)
18 Function dfsRevert(R,r):
19     for c ∈ children of r do
20         if c is n-vertex then
21             dfsRevert(R,c)
22         Revert o(c)
```

correspond. The figure shows that at two times during its course, the algorithm obtains a mode set μ_{32} for the final activity 32. The union of the non-dominated modes of these two sets gives the complete trade-off curve for the instance.

We conclude this subsection by illustrating the improvement in computational efficiency for the reduction tree compared to the reduction plan. For the illustration we again use the example instance from Figure 6, we assume that each activity has 2 modes. Figure 8 shows how the instance is solved with the reduction tree approach. As all reduction operations are listed in the figure, the reader can verify that a total of 52 operations are required to obtain the complete trade-off curve. Figure 9 shows how the instance is solved if the reduction tree is converted to a reduction plan with the same reduction operations. Again the total number of operations can be counted, and amounts up to a total of 78. For this small example, the computational effort is decreased by approximately 33%. For larger instances where more node reductions are required and activities have more modes, this effect can have a large impact on the computation time.

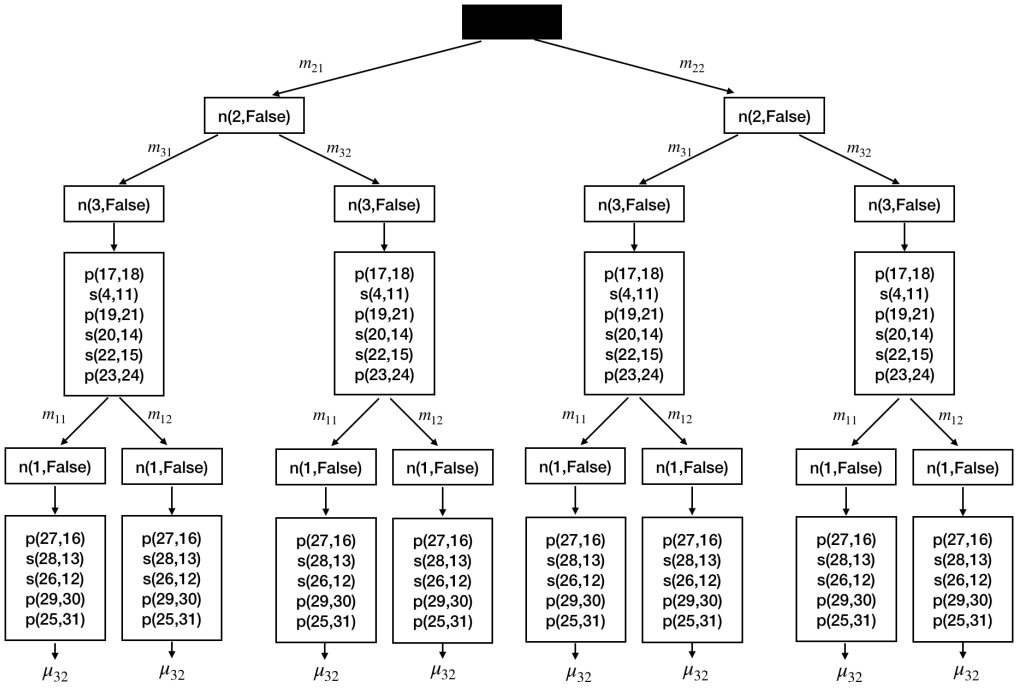


Figure 9: Example reduction plan solution algorithm

5.3. Extension to other objectives

The algorithm discussed in the previous subsection is suited for solving the curve problem, but it is generalizable to other objective functions. We start by

discussing adaptations for the budget and deadline variants of the DTCTP and follow up with generalizations to other cost types.

For the budget variant, two dominance rules can be applied during the search process, these were originally introduced by Demeulemeester et al. (1996). In the *resource-based rule*, a minimum cost is calculated by summing the cheapest modes of all activities and adding the additional cost for all the activities that were fixed to a more expensive mode by a node reduction. If this cost exceeds the given budget, this part of the search tree can be eliminated. In the *critical path-based rule*, the activities that were fixed by a node reduction are assigned the duration of that mode and all other activities are assigned the duration of their shortest, most expensive mode. If the critical path length of the resulting network is larger than the shortest feasible duration found during the search process, this part of the search space is dominated.

For the deadline variant, the same two rules can be used, but they need to be applied differently. If the minimum cost from the resource-based rule exceeds the lowest cost solution that has been found yet, this part of the search tree is dominated so the procedure can backtrack. The shortest duration from the critical path-based rule can be compared to the deadline, if it exceeds the deadline the procedure can backtrack.

The DTCTP can be generalized with additional costs, for instance *overhead* costs that are not related to the activities or a *penalty or bonus* if the project finishes too late or on time respectively. Our proposed algorithm is still applicable in these cases, because it outputs the trade-off curve with all non-dominated solutions. To incorporate overhead costs, the total project cost can be calculated as a sum of overhead costs and the mode cost of all activities. Furthermore, a penalty cost or bonus (i.e. cost reduction) can be added to the solutions on the trade-off curve that exceed or meet the deadline. On this modified curve, one can then select the optimal solution.

6. Algorithms

The computational performance of our exact procedure depends on the quality of the reduction tree that is constructed. This section is devoted to the algorithms for obtaining high-quality reduction trees. Each algorithm that we implemented consists of three components: a cost function to evaluate reduction trees, an algorithmic structure and a priority rule for selecting activities. We first discuss the cost function of a tree, which numerically expresses the quality of a tree. Then we describe the three algorithmic structures that we implemented. Last, we discuss the priority rules. At the end of this section we provide a final overview of the implemented algorithms.

6.1. Cost function

We quantify the quality of a tree R as a computational cost $c(R)$, which estimates the computational effort for the solution algorithm. Let $d(R)$ be the depth of the tree, i.e. the length of the longest path of n-vertices in R. As the

examples from the previous section showed, each level with n-vertices in the tree increases the computational effort by a multiplicative factor. In other words, the computational effort increases exponentially with the depth of the tree. We use two approaches to express the computational cost of a reduction tree. The *network approach* purely focuses on the number of required node reductions, while the *mode approach* also incorporates an estimate of the number of modes per node reduction. The cost $c(R)$ equals the cost of the root vertex $c(0)$, with the computational cost $c(r)$ of a vertex r being

$$c(r) = \begin{cases} f_r & \text{if } N(r) \text{ is empty ,} \\ f_r \cdot \sum\limits_{r' \in N(r)} c(r') & \text{if } N(r) \text{ is non-empty} \end{cases} \tag{1}$$

$N(r)$ is the subset of children of vertex r that are n-vertices and f_r is the cost factor of r. Note that this cost function only takes n-vertices into account.

In the network approach we focus on finding good reduction trees based on the network structure. Therefore, we set the cost factor $f_r = 2$ for any n-vertex. With this cost factor, $c(R)$ will be of the order $2^{d(R)}$, the cost function values trees with a lower depth. For the example tree of Figure 7, the cost is $2 \cdot (2 + 2) = 8$.

In the mode approach, f_r is set equal to $|\mu_i|$, the number of modes of the reduced activity. Assuming that the activities in Figure 7 all have three modes, the cost of this tree is $3 \cdot (3 + 3) = 18$. Similar to the first cost function, trees with a lower depth will have a lower cost. However, this cost function also gives preference to node reductions on activities with fewer modes.

Two of our algorithmic structures are based on the branch-and-bound methodology. These algorithms require lower and upper bounds for partial reduction trees, such that unpromising solutions can be quickly eliminated. We will now describe the lower and upper bounds for the network and mode cost function respectively.

6.1.1. Network cost bounds

As was stated in Section 3, the minimum number of node reductions for a graph D equals the size of the minimum vertex cover for its complexity graph $C(D)$. It is well-known that the number of vertices in a maximal matching $\mathcal{M}(C(D))$ of $C(D)$ provides a 2-approximation for the minimum vertex cover $\mathcal{V}(C(D))$ (Khot and Regev, 2008). It follows that $|\mathcal{M}(C(D))|/2 \leqslant |\mathcal{V}(C(D))|$, so it is a lower bound for the minimum vertex cover in $C(D)$ and thus for the minimum number of node reductions in D. This can now be used on the partially reduced nodes (i.e. L-nodes) in the reduction tree. For an L-node r with corresponding module M_r, a lower bound on the number of node reductions is $|\mathcal{M}(C(D) \cap M_r)|/2$. In the best case, all these node reductions could be executed independently from each other such that each of these reductions would be a child of the current L-node. The lower bound for this node is then obtained by the product of f_r and $|\mathcal{M}_r(C(D) \cap M_r)|/2$.

$$LB(r) = \begin{cases} f_r & \text{if } r \text{ is an n-vertex and } N(r) \text{ is empty ,} \\ f_r \cdot \sum_{r' \in N(r)} LB(r') & \text{if } r \text{ is an n-vertex and } N(r) \text{ is non-empty,} \\ f_r \frac{|\mathcal{M}(C(D) \cap M_r)|}{2} & \text{if } r \text{ is an L-vertex} \end{cases} \tag{2}$$

The calculation is illustrated on the partial reduction tree in Figure 10, which represents the instance $D|R(1)$ in Figure 7. The complexity graph $C(D|R(1))$ is shown on the right. For the left-most L-node, the maximal matching contains the nodes $\{c, f\}$, so the lower bound contains either c or f. The right-most L-node also contains 1 node in its lower bound, so the lower bound for this partial reduction tree is $2 \cdot (2 + 2)$.

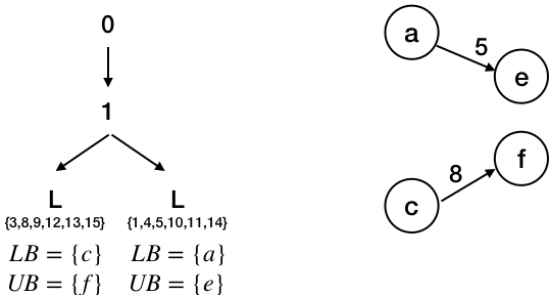


Figure 10: Example LB and UB calculation

For the upper bound, recall that applying a node reduction on each vertex in the vertex cover $\mathcal{V}(C(D))$ completely reduces the instance. It follows that for any L-node r with module M_r, applying the node reductions $M_r \cap \mathcal{V}(C(D))$ will completely reduce this module and result in a completely reduced tree. In the worst case, none of these reductions partition the module in submodules, such that all these n-vertices lie on one path in the tree. In this case, their costs need to be multiplied. This leads to the following upper bound:

$$UB(r) = \begin{cases} f_r & \text{if } r \text{ is an n-vertex and } N(r) \text{ is empty ,} \\ f_r \cdot \sum_{r' \in N(r)} UB(r') & \text{if } r \text{ is an n-vertex and } N(r) \text{ is non-empty,} \\ \prod_{i \in M_r \cap \mathcal{V}(C(D))} f_r & \text{if } r \text{ is an L-vertex} \end{cases} \tag{3}$$

In the example of Figure 10, the minimum vertex cover of the original complexity graph is $\mathcal{V}(C(D)) = \{e, f\}$. Therefore, the upper bound for this partial tree requires one node reduction in both of its L-nodes, so its upper bound is $2 \cdot (2 + 2)$.

6.1.2. Mode cost bounds

As this cost function takes into account the number of modes, a different approach for the lower bound is required. Each node in the complexity graph is assigned a weight, which is an estimate for the number of modes of the activity that corresponds to the node reduction. Note that for each node a forward or backward node reduction can be applied, which would lead to a different activity being reduced and thus a different number of modes. Therefore, we calculate a forward and a backward estimate for the nodes respectively. The calculation of these estimates is illustrated below. Figure 11 shows an instance with its corresponding complexity graph.

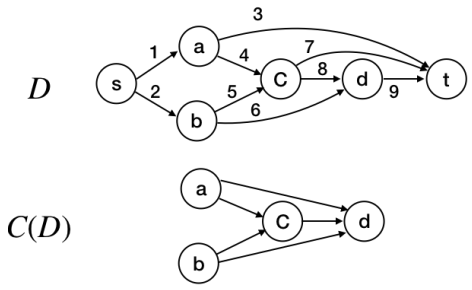


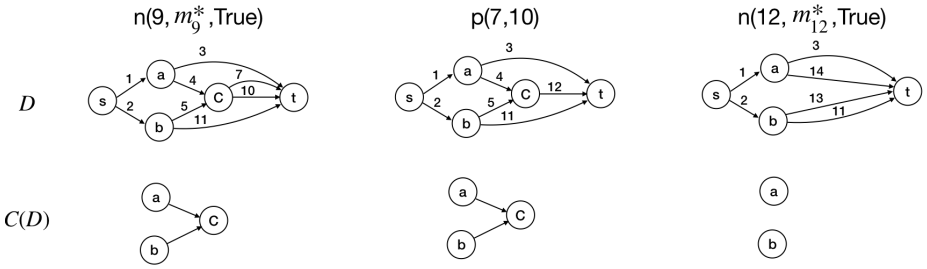
Figure 11: Example instance

Recall that a node can only be reduced in the forward (backward) direction if it has one outgoing (incoming) arc. Therefore, to obtain a cost estimate for certain node reductions, it may be necessary to first reduce other nodes. In the example, node c can only be reduced in the forward direction after the forward reduction of node d. Similar to the approach of Demeulemeester et al. (1996), the median mode (which we will denote by m_i^*) of an activity is used when a node reduction is applied on that activity i.

Figure 12 shows the sequence of reduction operations (starting from Figure 11) that are required to obtain a forward cost estimate for each node. Above each graph is the reduction operation that was applied to obtain the graph. The table below shows the cost estimate for each node. Note that after reducing nodes d and c, no node reductions need to be applied on a or b as the graph is already sp-reducible. The cost estimates for a and b are set to a value higher than any of the other costs as they cannot be reduced when applying forward reductions.

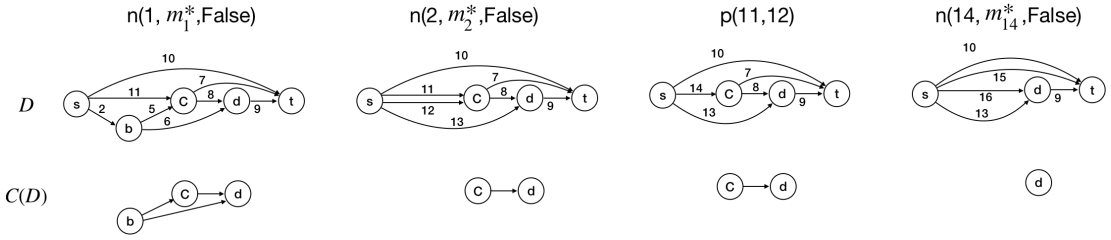
Figure 13 shows the approach for obtaining backward cost estimates. Note that in this approach, d cannot be reduced.

These cost estimates are used as node weights for the complexity graph. Now, the optimal set of node reductions is obtained by finding the minimum-weight vertex cover in the weighted complexity graph. For this problem, there also exists a 2-approximation method, the details are described by Bar-Yehuda and Even (1981). Let $\mathcal{W}(C(D))$ be the weight of this 2-approximation, then $\mathcal{W}(C(D))/2$ is a lower bound on the optimal solution. Note that we will cal-



Node	Cost
a	-
b	-
c	$\lvert\mu_{12}\rvert$
d	$\lvert\mu_9\rvert$

Figure 12: Forward cost estimate



Node	Cost
a	$\lvert\mu_1\rvert$
b	$\lvert\mu_2\rvert$
c	$\lvert\mu_{14}\rvert$
d	-

Figure 13: Backward cost estimate

culate two lower bounds, once with the weights from the forward approach $\mathcal{W}_f(C(D))/2$ (cfr. Figure 12) and once with those of the backward approach $\mathcal{W}_b(C(D))/2$ (cfr. Figure 13). Let $\mathcal{W}^*(C(D))/2$ be the minimum of these two. Now we can formulate the lower bound function for the mode cost approach:

$$LB(r) = \begin{cases} f_r & \text{if } r \text{ is an n-vertex and } N(r) \text{ is empty ,} \\ f_r \cdot \sum_{r' \in N(r)} LB(r') & \text{if } r \text{ is an n-vertex and } N(r) \text{ is non-empty,} \\ f_r \frac{\mathcal{W}^*(C(D) \cap M_r)}{2} & \text{if } r \text{ is an L-vertex} \end{cases} \tag{4}$$

It has the same recursive structure as the other lower bound (Equation 4), only the bound calculation of the L-vertices is different and f_r equals the mode estimate for the activity that is reduced at vertex r. The upper bound calculation for this cost function is the same Equation 3, with f_r set to the mode estimate.

6.2. Algorithmic structures

We implemented three structures for the search algorithms: a heuristic search, a breadth-first and a depth-first search. The heuristic search structure constructs one reduction tree. The outline of the heuristic is identical to Algorithm 1 of Subsection 5.1. In line 2 of the algorithm, an activity needs to be selected for the node reduction. This choice is determined by the priority rule, which will be discussed later.

The other two algorithmic structures are exact search procedures that find the optimal reduction tree, given a cost function. We propose two general approaches, a breadth-first (Algorithm 3) and a depth-first (Algorithm 4) search procedure. We will start by describing the breadth-first approach, which requires some additional notation. A subset S represents a set of activities that were reduced with node reductions. $\mathcal{S}_k$ contains all subsets of size k that were encountered by the algorithm. R^*_S is the best partial reduction tree (according to the cost function) that was found for the subset S. E_R is the set of tree vertices that are eligible, i.e. the vertices for which M_r does not contain smaller irreducible submodules.

For each k, the algorithm iterates over all subsets $S \in \mathcal{S}_k$. For a subset with optimal partial tree R^*_S, the procedure loops over all eligible vertices of that tree and for each vertex r it evaluates every possible node reduction in the corresponding module M_r. Given a reducible activity i in the module M_r, the procedure applies a node reduction on i and and all possible SP-reductions in M_r. These reductions result in a new partial reduction tree R', if $UB(R')$ is smaller than the currently optimal tree R^*, this optimal tree is updated. Then if $LB(R') < UB^*$, the current subset S' is created by adding the reduced activity i to S. Then the procedure updates the best tree for S' if R' has a lower UB than the current $R^*_{S'}$.

The second approach is a depth-first search approach. When it is called on a reduction tree R with corresponding set of node reductions S, the procedure

Algorithm 3: Breadth-first search

```
1  k ← 0
2  S_0 ← ∅
3  UB* ← ∞
4  while S_k is not empty do
5      for S ∈ S_k do
6          R ← R*_S
7          for Each r ∈ E_R do
8              for Each node-reducible activity i in M_r do
9                  R' ← apply n(i) and all possible SP-reductions in M_r
10                 if UB(R') < UB* then
11                     UB* ← UB(R')
12                     R* ← R'
13                 if LB_R' < UB* then
14                     S' ← S ∪ {i}
15                     if S' ∉ S_{k+1} then
16                         S_{k+1} ← S_{k+1} ∪ S'
17                     if UB(R') < UB(R*_S') then
18                         R*_S' ← R'
19     k ← k + 1
20 return R*
```

loops over all eligible tree vertices. It loops over each eligible vertex r and in each for loop applies a node reduction on one possible activity in M_r and all possible SP-reductions afterwards. Then, it evaluates the current tree R' in the same way as before. If the current solution is not dominated, the procedure calls itself on the new tree R' and new subset S'.

Algorithm 4: dfsOptimalTree(R, S)

```
1  for Each r ∈ E_R do
2      for Each node-reducible activity i in M_r do
3          R' ← apply n(i) and all possible SP-reductions in M_r
4          if UB(R') < UB* then
5              UB* ← UB(R')
6              R* ← R'
7          if LB_R' < UB then
8              S' ← S ∪ {i}
9              if UB(R') < UB(R*_S') then
10                 R*_S' ← R'
11                 dfsOptimalTree(R',S')
```

6.3. Priority rules

For the heuristic and depth-first search algorithms, the order in which activities are considered is important. We discuss two priority rules for the activity order. The first rule is proposed by Demeulemeester et al. (1996) and orders the activities in descending order of their index. We will call this rule MAXIND.

We propose a second rule (CON) that takes into account the connectivity of the complexity graph. An *articulation point* of a connected graph is a node such that deleting the node splits the graph in at least two separate connected components. We call a articulation point *non-trivial* if the graph remaining after its deletion contains at least two components that each have at least one arc. Let $V_1, \ldots, V_x$ be the vertex sets of the separate connected components after the deletion of the articulation point v, then the weight w_v of a node is $\prod_{1 \leqslant i \leqslant x} |V_i|$. If there are articulation points, the priority rule chooses the point with the maximum weight. If there are no articulation points, the rule selects the node with the highest number of adjacent nodes in the complexity graph. If there are multiple such nodes, let X be the set of these nodes. Then the rule selects the node that has the least adjacent nodes in X. Further ties are broken arbitrarily.

Figure 14 shows two complexity graphs with an illustration of the priority rule CON. The graph on the left-hand side contains multiple non-trivial articulation points, they are indicated by dotted lines. The weights of the nodes are also shown. For instance, deleting node c splits the graph in two components of 4 vertices, so its weight is 16. On the right-hand side there are no articulation

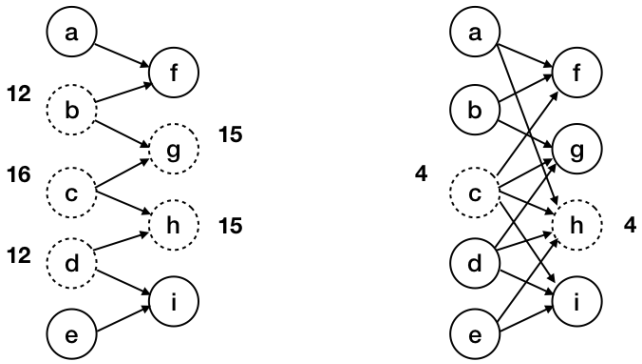


Figure 14: Examples priority rule

points, so the number of adjacent nodes is used. Nodes c and h both have 4 adjacent nodes. In this case, the rule would arbitrarily select one of these nodes.

6.4. Overview of algorithms

We implemented multiple algorithms for constructing reduction trees by using different options for the cost function, algorithmic structure and priority rule. Table 4 provides an overview. We denote the heuristic, breadth-first and depth-first structures by HEUR, BFS and DFS respectively. We implemented four exact algorithms (E1-E4) and one heuristic (H1). Preliminary experiments showed that in general (a) the mode cost function is more computationally expensive than the node cost function, (b) DFS is faster than BFS and (c) MAXIND is faster than CON. Based on these insights, we opted for four exact algorithms in order to avoid excessive computation times for obtaining a reduction tree. We also implemented one heuristic procedure, which uses the CON rule for selecting activities. Note that the cost function is irrelevant here as only one solution is constructed. For benchmarking purposes, we also reimplemented the two reduction plan approaches DHE1 and DHE2 from Demeulemeester et al. (1996). DHE1 constructs a reduction plan by solving a vertex cover (indicated by VC) problem on the complexity graph. DHE2 searches for the optimal reduction plan according to the mode cost function, using a depth-first search with the MAXIND priority rule.

7. Computational results

In this section we will compare the different proposed algorithms. The algorithms were tested on three different datasets. We generated the sets DTP20 and DTP30 with the activity-on-the-node generator RanGen2 (Vanhoucke et al., 2008). The sets contain 900 instances of 20 and 30 activities respectively. To control the network topology, we used the SP-indicator, which is a measure for the seriality of an instance. A completely parallel or serial network has a value

Algorithm	Cost function	Structure	Priority	Output
E1	Node	BFS	-	Reduction tree
E2	Node	DFS	MAXIND	Reduction tree
E3	Node	DFS	CON	Reduction tree
E4	Mode	DFS	MAXIND	Reduction tree
H1	-	HEUR	CON	Reduction tree
DHE1	-	VC	-	Reduction plan
DHE2	Mode	DFS	MAXIND	Reduction plan

Table 4: Overview of algorithms

of 0 or 1 respectively. The instances were then converted to the activity-on-the-arc format with the algorithm of De Reyck and Herroelen (1996). Although this algorithm may introduce additional dummy activities, it does not increase the complexity index of the instance. The set DTCTP98 was generated by Demeulemeester et al. (1998) and contains 1,800 instances of varying sizes. Table 5 summarises the three datasets: 'Size' indicates the number of instances, '$|N|$' the number of activities per instance, '$|\mu|$' the range of number of modes per activity and 'SP' the range of SP-values for the instances.

Set	Size	$\|N\|$	$\|\mu\|$	SP
DTP20	900	20	[2,4]	[0.1,0.9]
DTP30	900	30	[2,4]	[0.1,0.9]
DTCTP98	1,800	[10,50]	[2,11]	[0.14,0.44]

Table 5: Summary of the used datasets

All tests described in this section were performed on the STEVIN HPC-UGent Supercomputer (Processor architecture: 2x18-core Intel Xeon Gold 6140, clock speed 2.3 GHz). The memory limit was set to 50GB for each execution. In total, more than 2,000 hours of computation time have been devoted to the experiments.

7.1. DTP20 and DTP30

Our first set of experiments will be executed on the sets DTP20 and DTP30. In the first experiment we evaluate whether the reduction tree approach actually provides improvement over the reduction plan approach. Therefore, we compare the depth of the obtained reduction tree with the complexity index of the instance. If the depth is smaller than the CI, this indicates that the instance can be decomposed into smaller modules and using the reduction tree can provide computational improvements compared to the reduction plan approach.

Figure 15 shows the result of the first experiment for DTP30. The horizontal axis indicates the difference between the CI and the depth, the height of the bars indicates the number of instances within this category. The left-most figure shows that construction algorithm E1 obtains a reduction tree that improves upon the reduction plan for approximately one third of the instances. As E2

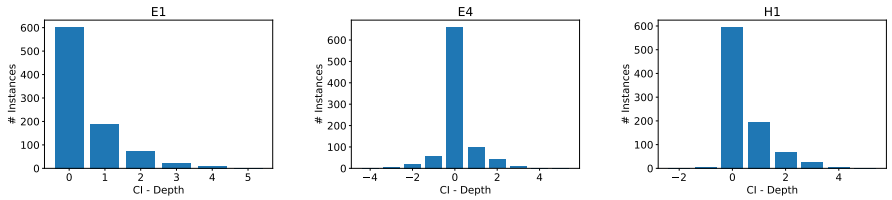


Figure 15: CI - depth for several algorithms on DTP30

and E3 optimize for the same objective as E1, their results are identical to E1. Interestingly, the figure for E4 shows that using the mode cost function may result in trees with both a lower and higher depth than CI. As E4 takes into account the estimated number of modes, the procedure sometimes constructs trees in which more node reductions will be required than the absolute minimum (i.e. CI). The results for the heuristic approach H1 show that it performs almost equally well in creating reduction trees as the exact approach E1.

We will now evaluate the computational performance of both the construction and solution algorithms. Table 6 reports the average computation times for the algorithms on DTP20 and DTP30. The column 'Prep time' shows the average time it takes to construct the reduction tree or plan, 'Sol time' exhibits the average time it takes to solve an instance, given the reduction plan or tree.

	DTP20		DTP30	
Algorithm	Prep time	Sol time	Prep time	Sol time
E1	0.09	0.01	113.12	7.95
E2	0.09	0.01	92.84	8.00
E3	0.09	0.01	106.68	8.12
E4	0.20	0.01	258.70	5.20
H1	0.00	0.01	0.00	7.18
DHE1	0.00	0.02	0.00	13.39
DHE2	0.02	0.01	12.28	10.78

Table 6: Average preparation and solution times

It is clear that DTP20 can easily be solved by all algorithms, with only small variations in computation times. Therefore, it is more interesting to compare the results on DTP30. We start by comparing the algorithms on their preparation times. The results show that for all exact construction algorithms (E1-E4), the preparation times are significantly larger than the solution times. Among the node-cost functions, E2 is the fastest, indicating that the depth-first search approach with the MAXIND priority rule is best suited. Using the mode-cost function increases the time to find the optimal tree significantly. This is mainly because the calculation of this cost function and its lower bound is more complex. DHE2 requires some time to construct its optimal reduction plan, but significantly less than our approaches. This indicates that constructing the optimal reduction tree is more complex than constructing the optimal reduction

plan. In contrast to the exact approaches, the heuristic H1 can almost instantaneously construct a reduction tree. DHE1 is also very quick, as it obtains a plan by solving a maximum matching problem.

From the perspective of the solution time, the reduction tree approach clearly outperforms the reduction plan approach. E4 performs the best, so incorporating mode-based information in the reduction tree is worth the effort. Nevertheless, the reduction tree approaches with the node-cost function (E1-E3) still outperform the reduction plan with a mode-cost function (DHE2). Interestingly, the heuristic H1 on average performs slightly better than E1-E3, even though it requires much less computational effort to construct a tree. From the perspective of the solution time, constructing the reduction tree with the mode-cost function (E4) is the best approach.

The most important evaluation criterion for algorithms is the total computation time. We therefore constructed Figures 16 and 17, which show the fraction of the instances where an algorithm is faster or slower than DHE2. If the difference is less than 0.01 second, the algorithms are considered to have equal computation times. The left-hand side reports the solution time, the right-hand side reports the total time, i.e. the sum of the preparation and solution time.

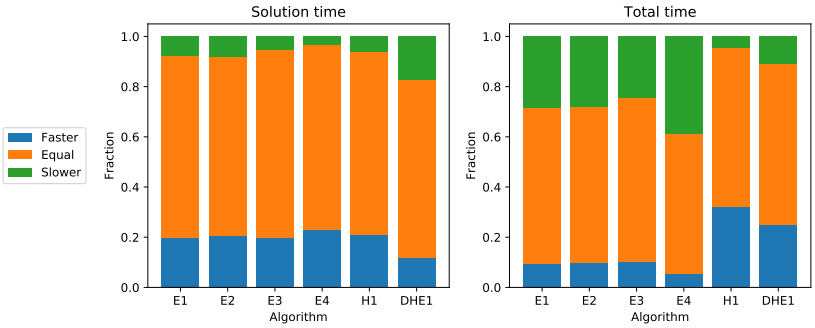


Figure 16: Comparison to DHE2 on DTP20

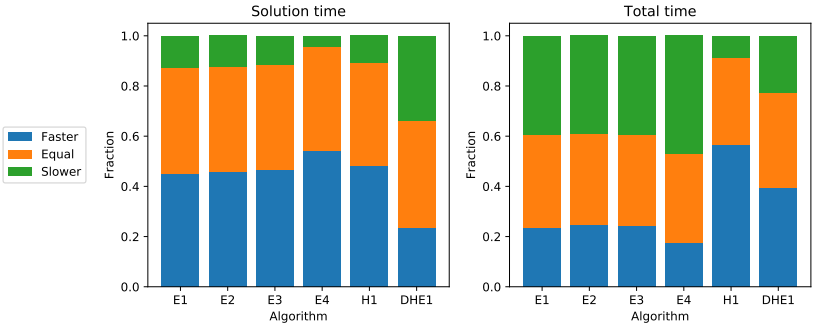


Figure 17: Comparison to DHE2 on DTP30

The figures confirm that in terms of solution times, E4 is the best algorithm. On DTP30, it outperforms DHE2 in 54% of the cases while being slower in only 4% of the cases. When looking at the total computation time, the heuristic approach H1 clearly stands out as the best performer. In 56% of the cases it outperforms DHE2, while being slower in 9%. Interestingly, when total time is considered the performance of E1-E4 is rather low, due to the long time it takes to construct the optimal reduction tree.

7.2. DTCTP98

In the previous experiments, all algorithms could solve the instances to optimality, as the instances are relatively small. The next set of experiments is executed on the DTCTP98 set, which contains larger instances with activities that have larger mode sets. It is likely that constructing the optimal reduction tree or reduction plan becomes untractable for the more difficult instances. In the first experiment we investigate how long it takes to construct an optimal reduction tree or plan. All construction algorithms were executed with a computation time limit of 30 minutes. Table 7 shows the results for the experiment, '% Found solution' and '% Optimal plan' report for which fraction of the instances the algorithm obtained a solution and the optimal solution respectively. 'Avg time' shows the average running time of the construction algorithm. '90-percentile' shows the time it takes to find the best solution achievable within 30 minutes for 90% of the instances. Note that this best solution is not necessarily the optimal solution.

Algorithm	% Found solution	% Optimal plan	Avg time	90-percentile
E1	100	69.28	633.37	38.80
E2	100	69.83	626.86	161.10
E3	100	69.28	633.82	432.00
E4	99.72	62.83	754.60	163.70
H1	100	-	0.00	0.00
DHE1	100	-	0.00	0.00
DHE2	82.00	75.11	238.46	2.00

Table 7: Construction reduction trees

A first observation is that DHE2 fails to obtain a plan for 18% of the instances due to excessive memory requirements. The same was observed with E4 for 0.28%. The table again confirms that constructing an optimal reduction plan (DHE2) is easier than constructing an optimal reduction tree (E1-E4), the average times are rather large. Furthermore, one again observes that constructing the optimal tree for the mode-cost function (E4) takes more time than for the node-cost function. The last column shows that using the 90-percentile as time limit would speed up the construction part significantly while keeping the quality of the reduction tree identical for the large majority of the instances.

In the next experiment we evaluate the capacity of the several algorithms to solve instances to optimality. Recall that the algorithm consists of two steps:

reduction tree or plan construction and trade-off curve calculation. For the sake of computational efficiency, we use the 90-percentiles from Table 7 as time limits for the first step. We set the time limit for the total algorithm to 1 hour, once a reduction tree or plan is constructed, the remaining time is used to calculate the trade-off curve. Table 8 shows the results of this experiment. The column '% Optimal' reports the fraction of instances for which the complete trade-off curve was calculated within the time limit. The last three columns respectively show the average time to construct the reduction tree or plan, the average time for computing the trade-off curve and the total run time.

Algorithm	% Optimal	Prep time	Sol time	Total time
E1	68.94	20.13	1,211	1,231
E2	70.95	74.83	1,116	1,191
E3	70.11	183.90	1,066	1,248
E4	70.78	81.95	1,104	1,186
H1	71.01	0.00	1,144	1,164
DHE1	65.59	0.00	1,337	1,337
DHE2	65.75	1.65	1,318	1,320

Table 8: Overview average computation times

We make several observations from the table. First, using the reduction tree approach allows to solve approximately 5% more instances than the reduction plan approach. Second, solving the problem with the truncated reduction trees is on average faster than using the truncated reduction plans. Interestingly, E3 performs best on solution time, showing that the priority rule CON performs well in finding good trees. From the total time perspective, the heuristic approach H1 performs best, closely followed by the mode-cost approach E4.

In a last analysis, we look at the percentage of instances for which an algorithm is faster or slower than DHE2. Figure 18 shows the comparison, both for the solution time and the total time. The cases are split in 4 categories. The blue and red categories respectively indicate the cases in which a certain algorithm is more than 1 second faster or slower than DHE2. Similarly, the orange and green categories respectively contain the instances for which the algorithm was at most 1 second faster or slower.

This figure confirms the previous observations and again shows that H1 is the best performing algorithm. In approximately 90% of the cases it is faster than DHE2, in 47% of the instances with more than 1 second.

The experiments of the previous two subsections show that the reduction tree approach provides improvements on the reduction plan approach. The best strategy is to quickly construct a reduction tree and to reserve the largest part of the solution time for calculating the trade-off curve.

7.3. Comparison to branch-and-bound procedure

Both the reduction tree and reduction plan approach are based on the concept of reducing an instance to one activity to obtain the complete trade-off

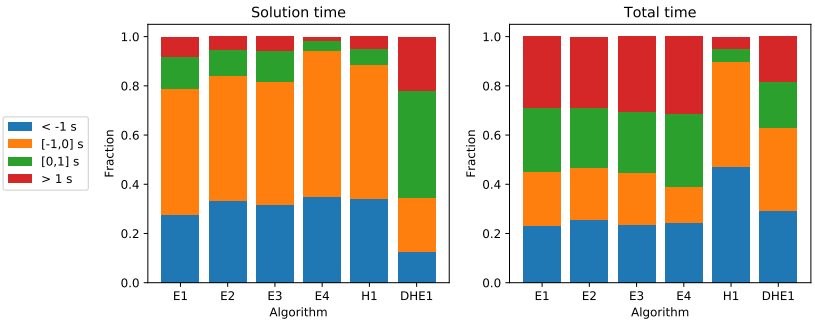


Figure 18: Comparison to DHE2 on DTCTP98

curve. In our last experiment we will compare the best performing reduction tree approach (H1) with the exact branch-and-bound algorithm from Demeulemeester et al. (1998), which tackles the problem in a completely different way. We received the source code for this algorithm from the authors. We denote this algorithm by D98 and tested it on the DTCTP98 dataset, with a time limit of 1 hour. The table below reports the fraction of the instances that the procedure could solve to optimality, the average computation time in seconds and the fraction of instances on which the algorithm was faster than the other one.

Algorithm	% Solved	Time	% Faster
H1	71.01	1,164	22%
D98	99.50	53	78%

Table 9: Comparison of H1 and D98

The results show that the D98 algorithm is better suited to solve the DTCTP as it is considerably faster and can solve almost all instances to optimality. However, it does not completely dominate H1, the reduction tree approach is still faster on 22% of the instances. To analyse the causes of the performance differences, we show the impact of different factors on the CPU-time of the algorithms in Figure 19. The top left figure shows the average CPU time of H1 and D98 in function of the depth of the tree that was obtained by H1. Up to a depth of 6, the performance of both algorithms is comparable. However, above this threshold, the CPU time of H1 increases dramatically in comparison to D98. This can be explained by the fact that the computation time for the reduction tree grows exponentially in the number of sequential node reductions that are applied (i.e. the depth of the tree). As D98 uses a different solution approach, it is less sensitive to changes in the number of node reductions.

We will now evaluate the impact of three other factors on the performance of the two algorithms: the instance size, the average number of modes per activity and the size of the Pareto front (i.e. the number of non-dominated solutions in the optimal solution). Note that for the last factor, we only consider the instances that were solved to optimality. The CPU-times of both algorithms

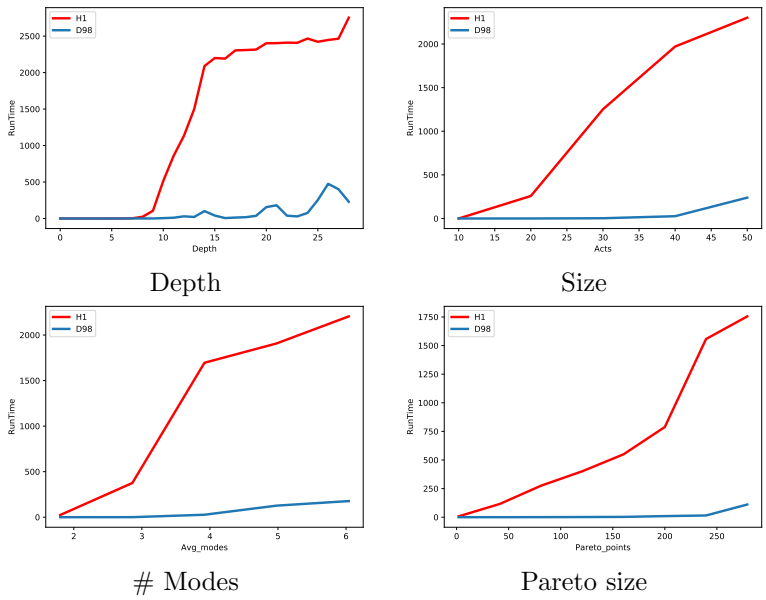


Figure 19: CPU time of H1 and D98 related to instance properties

increase when the instance becomes larger or when the average number of modes increases. These relations are not surprising, as both influence the number of possible solutions that need to be evaluated. However, it is clear that the CPU-time of D98 is less sensitive to changes in these factors than H1. The size of the optimal Pareto front shows a positive relation with the CPU-time, again the effect is stronger for H1. Two possible explanations can be given. First, the size of the Pareto front is positively correlated with the number of activities and the number of modes, which both have a positive effect on the CPU-time. Second, the time to compute a series or parallel reduction depends on the number of nodes that need to be considered. If the Pareto front becomes larger, it will take longer to make the intermediary computations of the algorithm.

We can conclude that the reduction tree approach is mainly competitive with the D98 when its tree depth is at most 6. The analysis for the other figures show that D98 in general is more scalable to more challenging instances. Nevertheless, the reduction tree approach is still valuable, as it shows that decomposing the instance into smaller modules by means of node reductions improves the running time of the solution algorithm. Further research could investigate the possibilities of a hybridisation of the solution procedures H1 and D98. In a first step, an instance could be decomposed in smaller modules by applying a small number of node reductions. In the second step, the remaining modules could then be solved independently with the more efficient algorithm D98. Future research could also investigate more advanced multipass heuristics or metaheuristics for constructing reduction trees.

8. Conclusion

In this paper we discuss a new exact solution approach for the curve problem of the DTCTP. This research is the first in the literature that incorporates modular decomposition in a solution procedure for this problem. We introduce the reduction tree, a data structure that exploits the modular decomposition structure of the network and we show how it is related to the complexity graph of the instance. We then elaborate on how the reduction tree is used to solve the problem by reducing the network to a single arc.

We propose heuristic and exact algorithms for constructing good reduction trees, using two objective functions and two activity priority rules. Our computational experiments show that using the reduction tree outperforms the reduction plan approach, an existing solution procedure that is also based on the concept of network reduction. Further analysis showed that the best strategy is to construct a reduction tree using a fast construction heuristic that takes into account the connectivity of the complexity graph and to allocate the largest part of the computation time to computing the complete trade-off curve. A comparison with a branch-and-bound procedure from literature reveals that the performance of the reduction tree decreases when more than 6 sequential node reductions are required.

Modular decomposition is a general framework that is not specific to one solution algorithm. Therefore, future research could investigate the potential of incorporating modular decomposition in other exact and heuristic procedures. Furthermore, one could investigate a hybridisation of the reduction tree approach and for instance the branch-and-bound procedure that was used in the experiments. A few well chosen node reductions could decompose the instance in smaller modules, which can then be solved by the more efficient branch-and-bound procedure.

Acknowledgements

We acknowledge the support provided by the Special Research Fund (BOF grant no. DOC014-18 Van Eynde) and the National Bank of Belgium for providing the first author with a pre-doctoral fellowship. The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, FWO and the Flemish Government – department EWI.

Agdas, D., Warne, D. J., Osio-Norgaard, J., and Masters, F. J. (2018). Utility of genetic algorithms for solving large-scale construction time-cost trade-off problems. *Journal of Computing in Civil Engineering*, 32(1):04017072.

Akkan, C., Drexl, A., and Kimms, A. (2005). Network decomposition-based benchmark results for the discrete time–cost tradeoff problem. *European Journal of Operational Research*, 165(2):339–358.

Aminbakhsh, S. and Sonmez, R. (2016). Discrete particle swarm optimization method for the large-scale discrete time–cost trade-off problem. *Expert Systems with Applications*, 51:177–185.

Aouam, T. and Vanhoucke, M. (2019). An agency perspective for multi-mode project scheduling with time/cost trade-offs. *Computers & Operations Research*, 105:167–186.

Bar-Yehuda, R. and Even, S. (1981). A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203.

Bein, W. W., Kamburowski, J., and Stallmann, M. F. (1992). Optimal reduction of two-terminal directed acyclic graphs. *SIAM Journal on Computing*, 21(6):1112–1129.

Chassiakos, A. P. and Sakellaropoulos, S. P. (2005). Time-cost optimization of construction projects with generalized activity constraints. *Journal of Construction Engineering and Management*, 131(10):1115–1124.

Coelho, J. and Vanhoucke, M. (2018). An exact composite lower bound strategy for the resource-constrained project scheduling problem. *Computers and Operations Research*, 93:135–150.

De, P., Dunne, E. J., Ghosh, J. B., and Wells, C. E. (1995). The discrete time-cost tradeoff problem revisited. *European journal of operational research*, 81(2):225–238.

De, P., Dunne, E. J., Ghosh, J. B., and Wells, C. E. (1997). Complexity of the discrete time-cost tradeoff problem for project networks. *Operations research*, 45(2):302–306.

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(2):347–366.

Değirmenci, G. and Azizoğlu, M. (2013). Branch and bound based solution algorithms for the budget constrained discrete time/cost trade-off problem. *Journal of the Operational Research Society*, 64(10):1474–1484.

Deineko, V. G. and Woeginger, G. J. (2001). Hardness of approximation of the discrete time-cost tradeoff problem. *Operations Research Letters*, 29(5):207–210.

Demeulemeester, E., De Reyck, B., Foubert, B., Herroelen, W., and Vanhoucke, M. (1998). New computational results on the discrete time/cost trade-off problem in project networks. *Journal of the operational research society*, 49(11):1153–1163.

Demeulemeester, E. L., Herroelen, W. S., and Elmaghraby, S. E. (1996). Optimal procedures for the discrete time/cost trade-off problem in project networks. *European Journal of Operational Research*, 88(1):50–68.

Elmaghraby, S. E. (1993). Resource allocation via dynamic programming in activity networks. *European Journal of Operational Research*, 64(2):199–215.

Erenguc, S. S., Tufekci, S., and Zappe, C. J. (1993). Solving time/cost trade-off problems with discounted cash flows using generalized benders decomposition. *Naval Research Logistics (NRL)*, 40(1):25–50.

Feng, C.-W., Liu, L., and Burns, S. A. (1997). Using genetic algorithms to solve construction time-cost trade-off problems. *Journal of computing in civil engineering*, 11(3):184–189.

Fulkerson, D. (1961). A network flow computation for project cost curves. *Management Science*, 7:167–178.

Grigoriev, A. and Woeginger, G. J. (2004). Project scheduling with irregular costs: complexity, approximability, and algorithms. *Acta Informatica*, 41(2-3):83–97.

Hafızoğlu, A. and Azizoğlu, M. (2010). Linear programming based approaches for the discrete time/cost trade-off problem in project networks. *Journal of the Operational Research Society*, 61(4):676–685.

Hazır, Ö., Erel, E., and Günalay, Y. (2011). Robust optimization models for the discrete time/cost trade-off problem. *International Journal of Production Economics*, 130(1):87–95.

Hazır, Ö., Haouari, M., and Erel, E. (2010a). Discrete time/cost trade-off problem: A decomposition-based solution algorithm for the budget version. *Computers & Operations Research*, 37(4):649–655.

Hazır, Ö., Haouari, M., and Erel, E. (2010b). Robust scheduling and robustness measures for the discrete time/cost trade-off problem. *European Journal of Operational Research*, 207(2):633–643.

He, Z., He, H., Liu, R., and Wang, N. (2017). Variable neighbourhood search and tabu search for a discrete time/cost trade-off problem to minimize the maximal cash flow gap. *Computers & Operations Research*, 78:564–577.

He, Z., Liu, R., and Jia, T. (2012). Metaheuristics for multi-mode capital-constrained project payment scheduling. *European Journal of Operational Research*, 223(3):605–613.

He, Z., Wang, N., Jia, T., and Xu, Y. (2009). Simulated annealing and tabu search for multi-mode project payment scheduling. *European Journal of Operational Research*, 198(3):688–696.

He, Z., Wang, N., and Li, P. (2014). Simulated annealing for financing cost distribution based project payment scheduling from a joint perspective. *Annals of Operations Research*, 213(1):203–220.

He, Z. and Xu, Y. (2008). Multi-mode project payment scheduling problems with bonus–penalty structure. *European Journal of Operational Research*, 189(3):1191–1207.

HE, Z.-w., LIU, R.-j., HU, X.-b., and Yu, X. (2009). Client perspective based multimode project payment scheduling problem and its heuristic algorithm. *Systems Engineering-Theory & Practice*, 29(2):70–77.

Herroelen, W. and De Reyck, B. (1999). Phase transitions in project scheduling. *Journal of the Operational Research Society*, 50(2):148–156.

Kamburowski, J., Michael, D., and Stallmann, M. (1992). Optimal construction of project activity networks. In *Proceedings of the 1992 Annual Meeting of the Decision Sciences Institute, San Francisco*, pages 1424–1426.

Khot, S. and Regev, O. (2008). Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349.

Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management science*, 41(10):1693–1703.

Leyman, P., Van Driessche, N., Vanhoucke, M., and De Causmaecker, P. (2019). The impact of solution representations on heuristic net present value optimization in discrete time/cost trade-off project scheduling with multiple cash flow and payment models. *Computers & Operations Research*, 103:184–197.

Li, H., Xu, Z., and Wei, W. (2018). Bi-objective scheduling optimization for discrete time/cost trade-off in projects. *Sustainability*, 10(8):2802.

Li, X., He, Z., Wang, N., and Vanhoucke, M. (2020). Multimode time-cost-robustness trade-off project scheduling problem under uncertainty. *Journal of Combinatorial Optimization*, pages 1–30.

Michael, D. J. and Kamburowski, J. (1993). On the minimum dummy-arc problem. *RAIRO-Operations Research-Recherche Opérationnelle*, 27(2):153–168.

Ning, M., He, Z., Jia, T., and Wang, N. (2017). Metaheuristics for multi-mode cash flow balanced project scheduling with stochastic duration of activities. *Automation in Construction*, 81:224–233.

Piper, C. (2005). Cpsim2: The critical path simulator (windows version). *Richard Ivey School of Business*.

Said, S. S. and Haouari, M. (2015). A hybrid simulation-optimization approach for the robust discrete time/cost trade-off problem. *Applied Mathematics and Computation*, 259:628–636.

Servranckx, T. and Vanhoucke, M. (2021). Essential skills for data-driven project management: A classroom teaching experiment. *Journal of Modern Project Management*, 8(4):123–139.

Skutella, M. (1998). Approximation algorithms for the discrete time-cost tradeoff problem. *Mathematics of Operations Research*, 23(4):909–929.

Sonmez, R. and Bettemir, Ö. H. (2012). A hybrid genetic algorithm for the discrete time–cost trade-off problem. *Expert Systems with Applications*, 39(13):11428–11434.

Szmerekovsky, J. G. and Venkateshan, P. (2012). An integer programming formulation for the project scheduling problem with irregular time–cost tradeoffs. *Computers & Operations Research*, 39(7):1402–1410.

Van Eynde, R. and Vanhoucke, M. (2022). A theoretical framework for instance complexity of the resource-constrained project scheduling problem. *Mathematics of Operations Research*.

Vanhoucke, M. (2005). New computational results for the discrete time/cost trade-off problem with time-switch constraints. *European Journal of Operational Research*, 165(2):359–374.

Vanhoucke, M. (2006). Work continuity constraints in project scheduling. *Journal of Construction Engineering and Management*, 132(1):14–25.

Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., and Tavares, L. V. (2008). An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, 187(2):511–524.

Vanhoucke, M. and Debels, D. (2007). The discrete time/cost trade-off problem: extensions and heuristic procedures. *Journal of Scheduling*, 10(4):311–326.

Vanhoucke, M., Demeulemeester, E., and Herroelen, W. (2002). Discrete time/cost trade-offs in project scheduling with time-switch constraints. *Journal of the Operational Research Society*, 53(7):741–751.

Vanhoucke, M., Vereecke, A., and Gemmel, P. (2005). The project scheduling game (PSG): Simulating time/cost trade-offs in projects. *Project Management Journal*, 36:51–59.

Wauters, M. and Vanhoucke, M. (2016). A study on complexity and uncertainty perception and solution strategies for the time/cost trade-off problem. *Project Management Journal*, 47(4):29–50.