FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

The Rooted Delay-Constrained Steiner Tree Problem with Uncertain Delays

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Roman Karl
Matrikelnummer 0825704

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Dipl.-Ing. Dr.techn. Markus Leitner
Proj.Ass. Dipl.-Ing. Dr.techn. Mario Ruthmair

Wien, 25.11.2013

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

The Rooted Delay-Constrained Steiner Tree Problem with Uncertain Delays

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Roman Karl
Registration Number 0825704

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Dipl.-Ing. Dr.techn. Markus Leitner
Proj.Ass. Dipl.-Ing. Dr.techn. Mario Ruthmair

Vienna, 25.11.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Roman Karl
Kimmerlgasse 2, 1110 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Abstract

The *rooted delay-constrained Steiner tree (RDCST) problem* is a combinatorial optimization problem. The task is to find a tree in a given weighted graph. The tree should have minimal weight which is defined as the sum of the edge weights. Furthermore, it should satisfy two constraints. The first is that the so-called terminal nodes have to be part of the tree. Additionally to the weight, every edge has also a given delay. The second constraint is that the overall delay on a path from a given root node to a terminal node stays within a given bound. This problem sometimes occurs when planning networks. For many services it is important that the delay between client and server does not get too high. A typical example are network applications with user interaction.

In optimization algorithms we usually assume that all input values are given precisely. But in practice these values are often affected by some kind of uncertainty. Inaccuracies occur inevitably with many measurements. Another source for uncertainty is data that is not yet present and therefore has to be predicted. Solutions of optimization problems can become infeasible because of the variability of input data. In practice this often means that the solution is of no use. Also the delays in a network are commonly affected by some jitter. We investigate for the RDCST problem how uncertainties can be incorporated into the optimization process.

We present algorithms based on mixed integer linear programming with which it is possible to find solutions of realistic instances of the optimization problem. These solutions feature a specific degree of robustness, which means that they stay feasible if actual values diverge from the assumed values. This degree can be adjusted accordingly to the respective requirements. The examined algorithms are exact. Thus, the best solution is found which fulfils the constraints. We present several ways of including uncertainties into the definition of the RDCST problem and its solution algorithms.

There are already methods to solve both the deterministic problem and general robust problems with integer linear programs. We show how both methods can be combined.

Kurzfassung

Das *Rooted Delay-Constrained Steiner Tree (RDCST) Problem* ist ein kombinatorisches Optimierungsproblem, bei dem ein Baum in einem gegebenen gewichteten Graphen gesucht wird. Dieser Baum soll ein minimales Gesamtgewicht haben, welches als die Summe der Kantengewichte definiert ist. Für den Baum gelten dabei zwei Nebenbedingungen. Die erste legt fest, dass die sogenannten Terminal-Knoten im Baum enthalten sein müssen. Zusätzlich zu den Gewichten werden für alle Kanten auch Übertragungszeiten definiert. Die zweite Nebenbedingung ist, dass die Gesamtübertragungszeit auf jedem Pfad zwischen dem gegebenen Wurzelknoten und einem Terminal-Knoten unter einer bestimmten Schranke liegen muss. Das Problem findet eine Anwendung bei der Planung von Netzwerken. Für viele Dienste ist es besonders wichtig, dass die Übertragungszeiten zwischen Client und Server nicht zu hoch werden. Typisch hierfür sind Netzwerk-Anwendungen mit Benutzer-Interaktion.

Bei Optimierungsalgorithmen geht man oft davon aus, dass alle Eingabewerte genau bestimmt werden können. In der Praxis kommt es allerdings häufig vor, dass diese Werte einer gewissen Unsicherheit unterliegen. Ungenauigkeiten entstehen zwangsläufig bei vielen Messungen. Andere Quellen für Unsicherheiten sind Daten, die erst in der Zukunft entstehen und davor nur geschätzt werden können. Lösungen von klassischen Optimierungsproblemen können durch die Schwankungsbreite der zugrunde liegenden Daten ungültig und aus diesem Grund in der Praxis mitunter gar nicht mehr verwendet werden. Auch die Übertragungszeiten in einem Netzwerk unterliegen häufig einer merkbaren Schwankung. Wir untersuchen anhand des RDCST Problems, welche Möglichkeiten zur Verfügung stehen, um Unsicherheiten in den Optimierungsprozess einzubeziehen.

Wir stellen Algorithmen basierend auf ganzzahliger linearer Programmierung vor, mit denen es möglich ist, Lösungen zu realistischen Instanzen des Optimierungsproblems zu finden. Diese Lösungen weisen eine gewissen Grad an Robustheit auf, was bedeutet, dass sie auch bei einer Schwankung der Werte gültig bleiben. Dieser Grad kann aufgrund der jeweiligen Anforderungen an die Lösungen variiert werden. Die behandelten Algorithmen sind exakt, finden also die beste Lösung, die alle Bedingungen erfüllt. Wir stellen einige Alternativen vor, wie die Unsicherheiten in die Definition des RDCST Problems und in die entsprechenden Algorithmen eingebunden werden können. Sowohl für das deterministische Problem als auch für allgemeine robuste Probleme stehen bereits Lösungsansätze im Bereich der ganzzahligen linearen Programmierung zur Verfügung. Wir zeigen, wie beide Ansätze kombiniert werden können.

Contents

CHAPTER 1

Introduction

1.1 Aim of Work

The focus of this thesis lies on the rooted delay-constrained Steiner tree (RDCST) problem which is a well studied combinatorial optimization problem. There are several algorithms to solve it, which can be useful when planning networks. In practice there are some limitations when applying these algorithms to problem instances. Often it is not possible to determine all parts of an instance like it is defined for the problem. The reason for this are uncertainties which are typically included in real world data. This thesis presents several ways for algorithms to operate with uncertainties, so that it is possible to find solutions for the considered problem.

1.2 Optimization with Uncertain Data

For an optimization problem it is often assumed that the information on its instances is complete. Many real world problems, however, do not have this property. If a problem instance contains measurement data, there is always an error, as it is known from physics. In many cases this error may be small enough so that it would not affect solutions to an optimization problem. If the error can be larger, it should no longer be ignored. At first, it is not clear how such inaccuracies should be considered in algorithms for the optimization problem. There is not only the risk of obtaining suboptimal solutions, but also the risk of a constraint violation. In many cases constraint violations cause solutions to be of no practical use. Therefore, an optimization algorithm sometimes has to be adapted before real world instances can be solved.

Besides imprecise measurements there are also other sources of uncertainty. It can be the case that information is not known at the time when a decision should be made. Let us consider the delay of network links. If we want to build a cheap network where some delay constraints should be satisfied, there is the problem that the delays can only be measured when the connection is already established. But there are no connections at the time when the network should be planned. So it is necessary to work with some predicted values. Without such a prediction

there is no data which can be used for optimization. But often data from the past can be used, or experts can come up with reasonable predictions.

A problem with complete information on its instances is usually called *deterministic problem*. It is clearly defined what optimal solutions are in its context. When uncertainties arise, it is not obvious anymore how optimality should be defined. The goal is to redefine the optimization problem so that a meaningful description of optimality follows. Such a problem is frequently called *robust problem* or *stochastic problem*, depending on the approach taken to handle the uncertainties, see also Sections 2.2 and 2.3. Such a redefinition includes a model of the uncertainties which in general does not perfectly coincide with the real world. This means that an optimal solution regarding a robust or stochastic problem is only optimal on this level of abstraction, and not necessarily in the real world. This is not surprising as we know that it is not possible to draw perfect conclusions from incomplete information. So solutions have properties with respect to the problem, the model and the real world. These properties can be different but should be similar.

There is a subtle difference between a robust and a stochastic problem. This is because there are two separate fields which deal with uncertain data in optimization problems which are *robust optimization* and *stochastic programming*. In robust optimization uncertain values are always assumed to lie within some given interval. Furthermore, it is usually assumed that there is no detailed information on the probability distribution of these values. Then, in general the worst case scenario is considered. A solution is called robust when it stays feasible in the model no matter what the real values of the uncertain data are. So a robust solution can only be infeasible in real world if the actual worst case is worse than the modelled worst case. Let us consider uncertain values which can get arbitrarily bad. This is a realistic scenario in many domains. For example, a problem instance could contain a prediction of the arrival time of a plane. The airline may state that the arrival time can diverge by at most one hour from the prediction which could also be a reasonable assumption at some level. This would mean that if earlier arrival times correspond to better cases, the worst case could be described with the predicted value plus one hour. But this is already a model, because with a very low probability the arrival can be much later. The worst case in the real world is that the plane does not reach its destination which can be defined as an infinite high deviation from the prediction. For robust optimization we have to define a finite worst case value. Even though very late arrivals occur only with a low probability, it can be seen that a definition of a worst case can be somewhat artificial. But this does not mean that robust optimization cannot be applied in such a scenario. Even if there is a discrepancy between the real world and the model, a robust approach can give good results.

In stochastic programming detailed information of the probability distributions of uncertain values is assumed to be given. The focus often lies on the expected case, but also other cases can be considered. The knowledge of properties of the probability distributions allows us to make stochastically well-founded statements about solutions. Such a statement can be that there is a probability of 10% that a specific solution is infeasible in the model. This should also be an accurate statement in real world, otherwise the application of stochastic programming is not reasonable.

The conversion to a robust or a stochastic problem is in general not problem specific. It depends more on the used approach of dealing with uncertainties. There is not only one for

robust optimization and another one for stochastic programming, but there are even many different approaches within each field. Many of them are presented together with its application on a general optimization problem. Also aspects of the implementation are often discussed. For a given robust or stochastic problem there are in general several different algorithms. This thesis considers only algorithms in the well-studied field of integer linear programming. The robust and stochastic variants of the problem described next allow some alternative formulations as *integer linear program (ILP)*, which makes them good candidates for analysing the formulations without being too problem specific.

1.3 The Rooted Delay-Constrained Steiner Tree Problem

The deterministic problem this work builds upon is a more general variant of the *Steiner tree problem*. We are given a weighted graph and the task is to find a connected subgraph with minimal weight. There are two different kind of nodes. Terminal nodes have to be part of the resulting subgraph, whereas potential Steiner nodes are only used if they reduce the weight. It can be shown that each optimal subgraph is indeed a tree if all edge weight are greater than zero. The generalisation to the *rooted delay-constrained Steiner tree (RDCST) problem* then works in the following way. Each edge has also an assigned delay and one of the nodes is the root node. Like a terminal node, the root node has to be part of a solution. Then there is the additional constraint that the cumulative delay from the root node to every other node has to stay below a given bound.

This is a typical network problem. Let us view the root node as a server and the other nodes as clients. The edges can be seen as cables, but they can even describe a whole link which can consist of cables, wireless connections and network devices. For such networks it is often important that the delay of transmissions does not get too high. Let us for example consider user interaction on a web application. Many users get easily frustrated if their computer does not respond for some time. For simple computations the major part of the delay comes from the communication over the network links when the server itself is not too busy. It can often be observed that the delay of the network links varies from time to time. There are many possible reasons for this like busy network devices or faulty cables. Hence, uncertain data arises very naturally.

It is worth noting that other parts of the problem instances could be affected by uncertainties, too. The edge weights can also be seen as costs for establishing this link. Such costs are also often not fixed at the time when the network is planned. They may be higher as predicted. Even more variations of the problem are possible if we consider incompleteness of the instance. As an example, it could be unknown whether a node is a potential Steiner or a terminal node in the beginning. This knowledge may be given after some links are already established. This induces a second planning phase but this time with complete information. Such problems are called two-stage problems. They are well studied in the field of stochastic programming.

This thesis only covers uncertain delays, because it is the most basic scenario. Only the constraints are affected by the uncertainty, which means that the task is to redefine feasibility of a solution. From this, a definition of optimality follows directly. Uncertain weights affect the objective function. So the feasibility of solutions does not change, but optimality has to

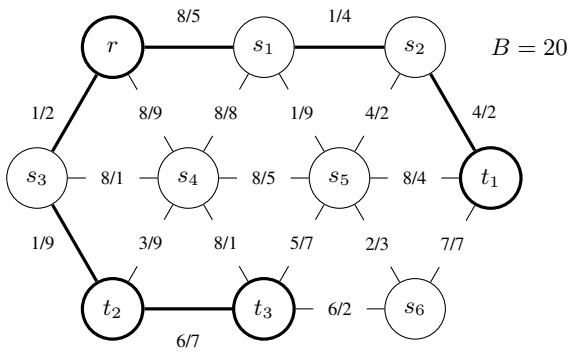


Figure 1.1: An instance of the RDCST problem and an optimal solution with weight 21. We have $S = \{s_i \mid i \in \mathbb{N}, 1 \leq i \leq 6\}$ and $T = \{t_1, t_2, t_3\}$. Each edge e is labelled with w_e/d_e and highlighted if it is part of the solution. The path to t_1 in the solution tree has a delay of 11 and the path to t_3 includes the path to t_2 and has a delay of 18.

be redefined. Sometimes, both scenarios can be treated similarly, but there are also notable differences. Two-stage problems diverge even further from the basic scenario.

The definition of the deterministic problem is given in Def. 1.1 and follows [44].

Definition 1.1 (Rooted Delay-Constrained Steiner Tree (RDCST) Problem) *We are given a graph $G = (V, E)$ and a delay bound $B \in \mathbb{N}$. The set of vertices V is partitioned into a root node r, a set of terminal nodes $T \subseteq V \setminus \{r\}$ and a set of potential Steiner nodes $S = V \setminus (T \cup \{r\})$. Each edge $e \in E$ has an assigned weight $w_e \in \mathbb{N}$ and a delay $d_e \in \mathbb{N}$.*

A feasible solution is a tree $G' = (V', E')$ with $T \cup \{r\} \subseteq V' \subseteq V$ and $E' \subseteq E$, fulfilling the delay constraints

$$d_{P_{G'}(r,t)} = \sum_{e \in P_{G'}(r,t)} d_e \leq B, \forall t \in T, \tag{1.1}$$

where $P_{G'}(r, t)$ is the edge set of the unique path from r to node t in tree G'. The task is to find a feasible solution with minimal weight

$$w_{G'} = \sum_{e \in E'} w_e. \tag{1.2}$$

Additionally, we define the set of arcs $A = \{(u, v) \mid \{u, v\} \in E, v \neq r\}$. A feasible solution can also be represented in a directed fashion as an arborescence $G' = (V', A')$ with r as its root and $A' \subseteq A$.

As a notational convenience d_{uv} and w_{uv} are used instead of $d_{\{u,v\}}$ and $w_{\{u,v\}}$, respectively.

Figure 1.1 shows a problem instance together with an optimal solution. As already mentioned, the RDCST problem is more general than the Steiner tree problem. The latter can be seen

as a special case of the RDCST problem where $B = \infty$. There are also other interesting special cases. For the *hop-constrained Steiner tree problem* [24] the only difference is that $d_e = 1$, $\forall e \in E$. A RDCST problem with $S = \emptyset$ is often called *rooted delay-constrained minimum spanning tree problem* [46]. Also often analysed in literature is the *hop-constrained minimum spanning tree problem* [15] which combines these two restrictions on the RDCST problem. Even for this special case the $\mathcal{NP}$-completeness can be shown. From this, the $\mathcal{NP}$-completeness of more general problems follows directly.

1.3.1 Related Work

The Steiner tree problem is a very old problem which dates back to the 19$^{\text{th}}$ century. It can be said that the first modern examination was done by Gilbert and Pollak in 1968 [22] where the relation to graph theory was discussed. Until then, the Steiner tree problem was never defined in the way, which is common nowadays. Also Gilbert and Pollak talked mainly about points in the Euclidean plane instead of nodes of a graph. In their definition terminal nodes were points with fixed coordinates whereas every possible point in the Euclidean plan was a potential Steiner node. This problem is nowadays referred to as *Euclidean Steiner tree problem* [18]. The first study of the Steiner tree problem in its graph theoretical form was done by Dreyfus and Wagner in 1971 [17].

The RDCST problem was introduced by Kompella et al. [33, 34]. An overview of its ILP formulations was given by Ruthmair [44]. One of them is a formulation that requires an instance transformation to a layered graph. A drawback of this approach is that the size of layered graphs increases with larger values of B. Therefore, Ruthmair and Raidl [45] presented an algorithm where the layered graph is not constructed completely at the beginning. Afterwards arcs are added if it is not possible to find an optimal solution with the reduced instance.

For the rooted delay-constrained minimum spanning tree problem there was some work done by Gouveia et al. [25]. They presented an ILP formulation with exponentially many variables and discussed how it can be solved effectively. One of their discussed algorithms is based on column generation. It was considered as ineffective until some improvements were done by Leitner et al. [36], see also [37]. Gouveia et al. used the term distance instead of delay which results in an alternative problem name. Also the RDCST problem is sometimes referred to as the *rooted distance-constrained Steiner tree problem*.

For the RDCST problem there were also several heuristics presented in the literature. Some of them are construction heuristics [1, 35]. Furthermore, metaheuristics, which include a genetic algorithm [56] and tabu-search [49], were applied.

1.4 Structure of the Work

In Chapter 2 basic concepts which are used in this work are explained. Chapter 3 then introduces a robust version of the RDCST problem. Thereby, we will stick to the approach which was presented by Bertsimas and Sim [6]. This leads to a model and a new problem definition. There are several ILP formulations presented which could be used to solve this problem. It is discussed that, from a stochastic point of view, this approach has some limitations. This motivates a

stochastic version of the RDCST problem in Chapter 4 which introduces a different model. It is pointed out that although robust optimization and stochastic programming are separate fields with different philosophies, they have a lot in common. Intuitively, a smaller instance should result in faster solving times. Therefore, Chapter 5 deals with reductions of the problem instances. A different way of dealing with uncertain delays is presented in Chapter 6. There an analysis whether one can avoid both previously introduced problems is given. The idea is to reuse algorithms for the deterministic problem and modify the problem instances instead. Some computational results are presented in Chapter 7 where not only robust and stochastic approach are compared, but also the algorithms based on the different formulations. Finally, Chapter 8 summarizes the thesis and discusses possible further work on the topic.

CHAPTER 2

Preliminaries

This thesis uses many concepts from integer linear programming like LP-based branch-and-bound, branch-and-cut and duality. For an explanation of these concepts the reader is referred to introductory literature [47,53]. The discussed optimization problems are mostly single-objective and a few are bi-objective. For simplicity, all single-objective optimization problems are assumed to be minimization problems.

2.1 Feasibility under Uncertain Data

When considering uncertain data in deterministic minimization problems there are the risks of suboptimality and infeasibility. In general, feasibility is more important than optimality. But if a solution should be feasible in every case, the value of the objective function can be very high. So it might be a good option to take a low risk of infeasibility if it decreases the objective function notably. Therefore, optimization with uncertain data often considers a specific case for which the optimization is done. The choice of the case allows control of the risk of infeasibility. The optimality of a solution is then redefined and depends on the chosen case. We distinguish the cases:

- Optimization with respect to a *good case* allows low values of the objective function. The probability that a constraint is infeasible can exceed 50%. A good case can be near or far from the expectation. If a solution is only valid for a good case far from the expectation, there is a high risk of a constraint violation. Optimization with respect to a good case is not very common.

- The *expected case* considers an average scenario. Therefore, each constraint has to be feasible for the expected value. For symmetric probability distributions it holds that if a solution is feasible for the expected case, every constraint is valid with a probability of at least 50%.

- For symmetric probability distributions it holds that if a solution is feasible for a *bad case*, every constraint is valid with a probability of more than 50%. There are a lot of bad cases including the worst case and cases near the expectation.
- The *worst case* considers the worst possible scenario. A solution is feasible for the worst case iff it is feasible for all possible values. Worst case optimization is the most conservative form of optimization with uncertain data.

2.2 Robust Optimization

In general the goal of robust optimization is to find an optimal solution for a given problem, taking all possible values for some uncertain data into account. The solution is then called robust, because it is valid even in the worst case.

Let us now consider a general minimization problem of the following form:

$$\min \sum_{j \in J} c_j x_j \tag{2.1}$$

$$\text{s.t.} \sum_{j \in J} a_{ij} x_j \leq b_i \qquad \forall i \in I \tag{2.2}$$

$$l_j \leq x_j \leq u_j \qquad \forall j \in J \tag{2.3}$$

For every j which lies is some set J there is a variable x_j. All other values are constants. Let us assume that a_{ij} is only a prediction of a random variable $\tilde{a}_{ij}$ which is the source of uncertainty. We assume that $\tilde{a}_{ij}$ has an unknown symmetric distribution and can only take values in the bounded interval $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$. This model of uncertainty makes it easy to optimize the worst case. Such a formulation was given by Soyster [50]. From the symmetric distribution follows, that a_{ij} is now the expected value of $\tilde{a}_{ij}$. Because of the linearity of the expectation operator, the linear program above gives us also an optimum for the expected case.

So if we wanted to consider only the expected case, there would be no need to extend the formulation. Therefore this is not a very interesting case in robust optimization. On the other hand the worst case is indeed something we want to consider, but the probability that it actually occurs is often very low. Let us call something a bad case if it lies between the expected and the worst case. The question arises how solutions can be found for at least some of these bad cases. There are different approaches for this kind of optimization. One of them, which is particularly interesting from a computational point of view, was introduced by Bertsimas and Sim [6] and is described in the next section.

By describing the interval for the random variables a model was created which does not necessarily coincide with the real world. The worst case can then be defined as the case where the minimal value of the objective function would be maximal for all possible results of the random variables. There are also other possible models. The scenario-based model was studied in detail by Mulvey et al. [42]. In this model possible values for a random variable are not given as an interval, but as a set of values that were observed in the past. An assignment for all random variables to one of its observed values is then called scenario. If it is assumed that

the random variables are not independent, a scenario should only consist of values that were observed together. It is an advantage over the interval-based model that it can be applied if there are dependent random variables. For the case of independence the interval-based model is more suited. One reason for this is that for a scenario-based model the instance can be very large if it contains a lot of observations.

2.2.1 The Approach of Bertsimas and Sim

For the expected case we have set zero random variables to their worst case value, whereas for the worst case it were all of them. The idea in this approach is to introduce a parameter Γ to control the number of random variables which are set to the worst value. The others stay at the expected case. The following formulation describes the robust problem:

$$\min \sum_{j \in J} c_j x_j \tag{2.4}$$

$$\text{s.t.} \sum_{j \in J} a_{ij} x_j + \max_{\substack{\{S_i \cup \{t_i\} | S_i \subseteq J_i, \\ |S_i| = \lfloor \Gamma_i \rfloor, t_i \in J_i \setminus S_i\}}} \{\sum_{j \in S_i} \hat{a}_{ij} y_j + (\Gamma_i - \lfloor \Gamma_i \rfloor) \hat{a}_{it_i} y_t\} \leq b_i \qquad \forall i \in I \tag{2.5}$$

$$-y_j \leq x_j \leq y_j \qquad \forall j \in J \tag{2.6}$$

$$l_j \leq x_j \leq u_j \qquad \forall j \in J \tag{2.7}$$

$$y_j \geq 0 \qquad \forall j \in J \tag{2.8}$$

If x_j can be negative, the variable y_j is necessary to get its absolute value. The interesting new part is the maximum term in constraints (2.5). Note that there can be a different value of Γ for each inequality and that it is not limited to integral numbers. The elements in S_i are those which take the worst value, because they have the highest $\hat{a}_{ij} y_j$. The element t_i is used for the fractional part of Γ_i and its random variable is assigned with a bad value.

This approach is well suited for implementations, because it is possible to transform the formulation above into a linear program. Such a transformation is done in Section 3.2.1.

Note that there is an alternative perspective for this approach. In this thesis different values of Γ are viewed as differently bad cases. That means that there is still a chance that a robust solution is not feasible for results of the random variables. Alternatively, it could be stated that it is only possible that at most Γ random variables take their worst value while the others take the expected value. Then we would have to speak just of the worst case but for different models. This perspective fits better to the philosophy of robust optimization. However, it is unrealistic that such a model coincides with the real world. It is hard to imagine an example, where a set of random variables behaves like that. So, if we abandon the idea of analysing the worst case, there has to be the risk that a robust solution is infeasible. But then also the question arises how likely this is. There cannot be a precise probability of feasibility, but Sim [48] provided some probabilistic bounds.

2.2.2 Related Work

Another robust approach for the interval-based model was introduced by Ben-Tal and Nemirovski [3, 4] and independently by El Ghaoui et al. [19, 20]. Even though this approach belongs to robust optimization, it has an interesting stochastic foundation, which is discussed in Section 4.1.

Often it is easier to investigate and discuss properties of a robust approach when focusing on a concrete problem instead of working with a generic optimization problem only. This allows also more practice-oriented statements concerning the performance and the applicability of robust approaches. One problem that was already considered very often is the spanning tree problem. It was studied with an interval-based model by Yaman et al. [54]. In their work the uncertainty affects the edge weights. This means that the uncertainty has to be handled together with the objective function. For the RRDCST problem the uncertainties occur in the inequalities. This is sometimes referred to as uncertainty associated with *hard constraints*. Both variants can often be treated similarly, but for some problems it happens that one of them is a lot easier to solve. Because robust optimization got most of its input quite recently, their study, published in 2001, can already be counted to old work.

The approach of Bertsimas and Sim was applied to the prize-collecting Steiner tree problem by Álvarez-Miranda et al. [2]. This is a variant of the Steiner tree problem where every node has some given profit and every edge has some given cost. There are no terminal nodes anymore which have to be part of a solution. Nodes are only part of an optimal solution if they help increasing the profit. Álvarez-Miranda et al. defined a robust counterpart of the problem where both profits and costs are assumed to be uncertain. An algorithm based on branch-and-cut was developed to solve it.

Robust variants are also investigated for other problems. The robust travelling salesman problem was studied by Montemanni et al. [41] who also worked with an interval-based model. A robust variant of the shortest path problem was introduced by Yu and Yang [55]. A robust knapsack problem was introduced by Bertsimas and Sim [6] as a demonstration of their approach. They assumed the weights to be uncertain and that, analogously as for the α-RRDCST problem from Chapter 3, a constant factor α exists which characterises the maximal variation of all item weights. The more general variant without the factor α was studied by Monaci et al. [40]. They used dynamic programming as it is often done to solve the deterministic knapsack problem.

Another problem, which is studied in several recent publications, is the robust capacitated vehicle routing problem. For this, the importance of considering uncertainties can be seen from practical examples. Gounaris et al. [23] defined the customer demands as random variables. The routes may be planned for a longer period whereas the demands of some costumer probably change several times. By only solving the deterministic problem, an expensive rebuild of the routes can be the consequence when a demand changes. Their work also discusses the relationship to a stochastic variant of the problem.

The approach of Bertsimas and Sim had a great influence in the field of robust optimization. But it has also some limitations. The assumption of a symmetric probability distribution, for example, does often not hold in real world. There are several studies on similar but more flexible approaches. One of this approaches considers an extended model of the uncertainties. The

so called *multi-band uncertainty* allows to incorporate different probability distributions. On overview on this topic was given by Büsing and D'Andreagiovanni [9]. Such approaches can also be categorized to stochastic programming.

2.3 Stochastic Programming

The main idea of stochastic programming is the same as in robust optimization. There is some source of randomness we have to consider in the optimization. But instead of focusing on the worst case, the target is to find solutions which are optimal for some case where we know important probabilistic properties. Because of computational difficulties many stochastic problems just consider the expected case. An expected case analysis for a RDCST problem with uncertain delays is very easy if expected edge delays are given. It is more difficult for the other cases. Each case has some dedicated probability p. If an optimization is done with respect to a given case, each constraint is not violated with a probability of at least p. It is necessary to have detailed information of the probability distribution of the random variables to achieve this goal, especially expected value and standard deviation.

2.3.1 Related Work

One of the first studies to stochastic programming was done by Dantzig in 1955 [16]. Since then it evolved to a wide field which is also the topic of several books. Introductions to the topic are given amongst others by Birge and Louveaux [7] and by Kall and Wallace [31]. Stochastic programming splits up into expected case analyses and *chance-constrained* programming.

The expected case analyses are often used in combination with two-stage problems. In the context of Steiner tree problems such an analysis was done by Gupta and Pál [27]. In the first stage they assumed that the set of terminal nodes is unknown, and only a probability of being a terminal node is given for each node. The second stage then reveals the set of terminal nodes. The problem of finding a solution with minimal cost gets hard when the prices increase in the second stage. It can be a good strategy to include edges to the solution in the first stage even if there is the risk that they might not be needed. The problem was also studied by Ljubić et al. [8]. Their main concern was to create a faster algorithm.

Two-stage problems usually tackle the expected case only, because working with quantiles instead of expected values can be much more complicated for such complex problems. More different cases are often analysed if it is possible to calculate quantiles effectively. These calculations can then be included into the constraints of the problem formulation which is also done in Chapter 4. Such constraints are called chance constraints. They were introduced by Charnes and Cooper [12]. Ishii et al. [30] applied these concepts to the spanning tree problem. It was also applied to a variant of the problem by Ishii et al. [29] where the task is to minimize the maximal weight of one edge. Interestingly both problems are equivalent for their deterministic versions which does not hold for the stochastic problems. This also indicates that a lot of variations of a problem are possible in stochastic programming.

Robust problems are often easier to solve but stochastic problems often provide a better definition of optimality. This inspired the idea of developing robust algorithms that provide a

good approximation to stochastic problems with chance constraints. Similar approaches for this were presented by different authors [10, 13, 21, 43]. The concept was applied to the knapsack problem by Klopfenstein and Nace [32].

Another class of problems was defined by Thiele [51] under the name *robust stochastic programming*. In robust optimization all probability distributions are assumed to be unknown whereas in stochastic programming there is complete knowledge of them. The assumption in robust stochastic programming lies somewhere in between. Probability distributions are assumed to be given, but they are uncertain. This means that the uncertainty occurs on two levels. First, a value itself is uncertain. The degree of uncertainty is slightly decreased by a probability distribution, which also is affected by uncertainty. Liu [38] stated that probability distributions are often not known in practice. Instead, experts come up with some guesses. Such guessed probability distributions often have a higher variance than the unknown real probability distributions. Liu coined the term *uncertain programming* which is founded in a comprehensive new theory.

2.4 Bi-Objective Optimization

Most optimization problems have exactly one objective function which should be minimized or maximized. In practice it is often not desirable to express the value of a solution with only one function. In *multi-objective optimization* there can be several objectives where each objective function should either be minimized or maximized. If the objectives are conflicting, there may be not only one optimal solution for a problem instance, but there can be a lot of solutions where it is not obvious how to rank them. A very common classification of solutions is *Pareto optimality*.

Definition 2.1 *A solution $w = (w_1, w_2, ..., w_n)$ of a multi-objective minimization problem is Pareto optimal if there is no other solution $w' = (w'_1, w'_2, ..., w'_n)$ with*

$$\forall i, 1 \leq i \leq n : w'_i \leq w_i \qquad \wedge \qquad \exists i, 1 \leq i \leq n : w'_i < w_i.$$

This thesis only considers optimization problems with at most two objectives. An optimization problem with two criteria is called *bi-objective* optimization problem. A solution of a bi-objective optimization problem is Pareto optimal if there is no other solution where one criterion is better and the other one is at least equally good. A solution is in the *Pareto front* if there is no other solution where both criteria are better. The Pareto front therefore contains all Pareto optimal solutions.

One method of finding all Pareto optimal solutions of a bi-objective problem instance is the *epsilon-constraint method*. A detailed explanation was given by Chankong and Haimes [11]. Let us consider a bi-objective minimization problem of the following form:

$$\min \left(f_1(\mathbf{x}), f_2(\mathbf{x})\right) \tag{2.9}$$

$$\text{s.t. } \mathbf{Ax} \leq \mathbf{b} \tag{2.10}$$

One solution of the Pareto front can be found by solving the following optimization problem.

$$\min f_1(\mathbf{x}) \tag{2.11}$$
$$\text{s.t. } \mathbf{Ax} \leq \mathbf{b} \tag{2.12}$$
$$f_2(\mathbf{x}) \leq c \tag{2.13}$$

Every choice for the constant c can result in a different solution of the Pareto front. All Pareto optimal solutions can then be iteratively found with the following steps.

1. c is initialized with ∞
2. The single-objective problem is solved. Its result is either a new solution $\mathbf{x}'$ of the Pareto front or no solution if no one exists.
3. If solution $\mathbf{x}'$ exists, $f_2(\mathbf{x}') - \epsilon$ is assigned to c, where ϵ is a sufficiently small value greater than zero, and it is continued with 2.

CHAPTER 3

The Robust Rooted Delay-Constrained Steiner Tree Problem

In Section 1.3 the RDCST problem was defined. But in the context of uncertain delays the definition of optimality has to be changed. A solution which has an undesired high probability of being infeasible should not be referred to as optimal anymore. This means that there has to be a new problem definition. We will stick to the approach of Bertsimas and Sim which was discussed together with basic concepts of robust optimization in Section 2.2.

Definition 3.1 *For a set of edges P, where each edge has a delay variation $\hat{d}_e \in \mathbb{Q}^+$, and a parameter $\Gamma \in \mathbb{Q}^+$ we define the delay variation $\hat{d}_P^\Gamma \in \mathbb{Q}^+$ as:*

$$\hat{d}_P^\Gamma = \begin{cases} \max\{\sum_{e\in F} \hat{d}_e + (\Gamma - \lfloor\Gamma\rfloor)\hat{d}_f \mid F \subset P, |F| = \lfloor\Gamma\rfloor, f \in P \setminus F\} & \text{if } |P| > \Gamma \\ \sum_{e\in P} \hat{d}_e & \text{otherwise} \end{cases} \tag{3.1}$$

Definition 3.2 (Robust Rooted Delay-Constrained Steiner Tree (RRDCST) Problem) *We are given a graph $G = (V, E)$, a delay bound $B \in \mathbb{N}$ and a parameter $\Gamma \in \mathbb{Q}^+$. The set of vertices V is partitioned into a root node r, a set of terminal nodes $T \subseteq V \setminus \{r\}$ and a set of potential Steiner nodes $S = V \setminus (T \cup \{r\})$. Each edge $e \in E$ has an assigned weight $w_e \in \mathbb{N}$, an expected delay $d_e \in \mathbb{N}$ and a maximal delay variation $\hat{d}_e \in \mathbb{Q}$ with $0 \leq \hat{d}_e < d_e$.*

A feasible solution is a tree $G' = (V', E')$ with $T \cup \{r\} \subseteq V' \subseteq V$ and $E' \subseteq E$, fulfilling the delay constraints

$$d^\Gamma_{P_{G'}(r,t)} = \sum_{e\in P_{G'}(r,t)} d_e + \hat{d}^\Gamma_{P_{G'}(r,t)} \leq B, \forall t \in T, \tag{3.2}$$

where $P_{G'}(r, t)$ is the edge set of the unique path from r to node t in tree G'.

The task is to find a feasible solution with minimal weight

$$w_{G'} = \sum_{e\in E'} w_e. \tag{3.3}$$

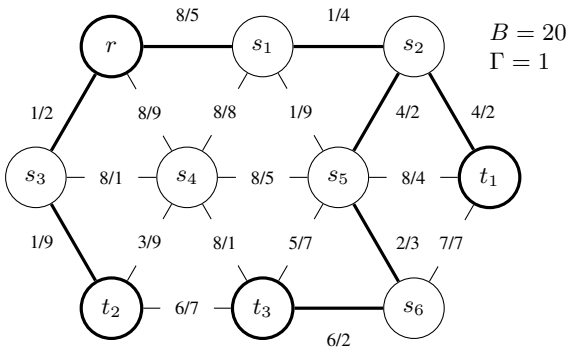


Figure 3.1: An instance of the 0.5-RRDCST problem and an optimal solution with weight 27. We have $S = \{s_i \mid i \in \mathbb{N}, 1 \leq i \leq 6\}$ and $T = \{t_1, t_2, t_3\}$. Each edge e is labelled with w_e/d_e and highlighted if it is part of the solution. A path P_i is the path from r to t_i in the solution tree. Then we have $d^{\Gamma}_{P_1}$ = 13.5, $d^{\Gamma}_{P_2} = 15.5$ and $d^{\Gamma}_{P_3} = 18.5$.

If the links of a network use the same techniques, their delays may have similar deviations. In this case an intuitive assumption is that the maximal delay variation of an edge is proportional to its expected delay. This leads to the following definition:

Definition 3.3 *The* α-RRDCST problem *is a special case of the RRDCST problem where a constant* $\alpha \in \mathbb{Q}$, $0 < \alpha < 1$, *exists such that* $\hat{d}_e = \alpha \cdot d_e, \forall e \in E$.

Figure 3.1 shows a problem instance together with an optimal solution. This can be compared to Fig. 1.1 which presents a solution for the deterministic problem. It can easily be seen that the RDCST problem is equivalent to the RRDCST problem with $\Gamma = 0$. When Γ is increased to 1, the paths to t_1 and t_2 stay the same, while the connection to t_3 changes. The old path P_o would have a total delay $d^{\Gamma}_{P_o}$ of $18 + 4.5 = 22.5$ and would therefore exceed the delay limit.

It is worth noting that it would be a different problem if we defined a solution not as a tree, but as a connected graph, as it is often done for the Steiner tree problem. In Fig. 3.2 there is a solution to an instance which is not feasible, because it violates the tree property. With an alternative definition the solution could be feasible, since only two edges can take their worst value $d_e + \hat{d}_e$. For all possible cases there remains one path to t which satisfies the delay constraint. If we accepted the solution as feasible, it would also be optimal. But the definition which is used in this thesis implies that edge $\{r, t\}$ is the only feasible and therefore optimal solution to the instance from Fig. 3.2a.

3.1 Limitations of the Robust Approach

With the approach of Bertsimas and Sim detailed information of the random variables is not needed which is useful if there is not much information of the probability distributions available.

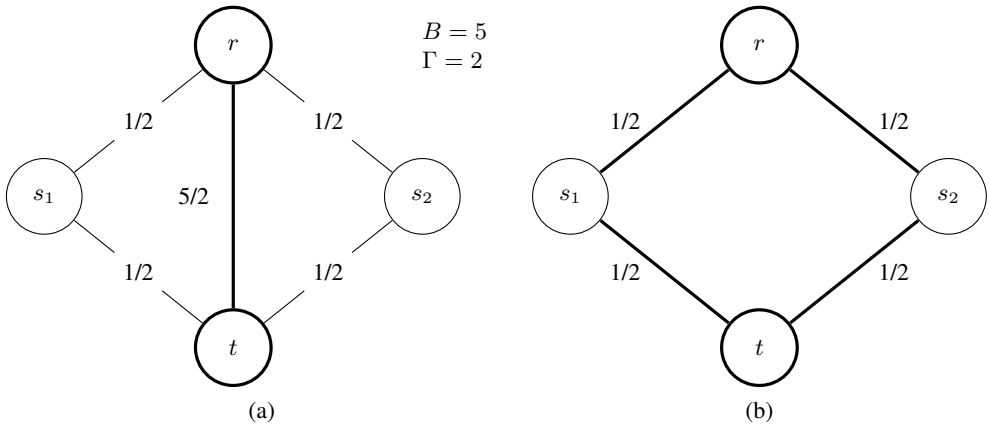


Figure 3.2: (a) An instance of the 0.5-RRDCST problem and an optimal solution with weight 5. Each edge e is labelled with w_e/d_e. We have $S = \{s_1, s_2\}$ and $T = \{t\}$. (b) An infeasible solution that has lower weight but violates the tree property.

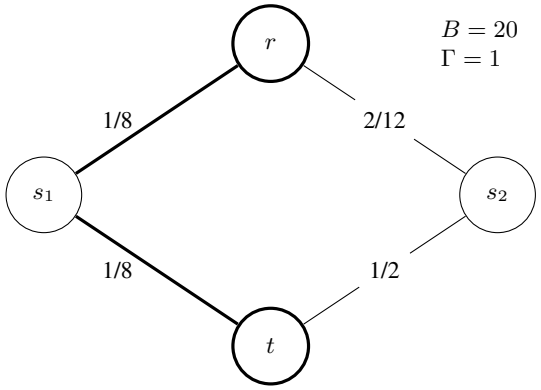


Figure 3.3: An instance of the 0.5-RRDCST problem and an optimal solution with weight 2. The promising solution with edges $\{\{r, s_2\}, \{s_2, t\}\}$ is deemed as suboptimal for every value of $\Gamma \leq 1$. We have $S = \{s_1, s_2\}$ and $T = \{t\}$. Each edge e is labelled with w_e/d_e.

This means that the goal is to find good results for arbitrary probability distributions. However, the optimal solutions may be not satisfying for some instances.

Let us consider Fig. 3.3 as an example. There we have only two possible feasible solutions which consists of paths $P_1 = \{\{r, s_1\}, \{s_1, t\}\}$ and $P_2 = \{\{r, s_2\}, \{s_2, t\}\}$, respectively. Trying all values of Γ will give us either no solution (for $\Gamma > 1$) or the solution consisting only of P_1 (for $\Gamma \leq 1$), because it has minimal weight. For the RRDCST problem both paths are deemed to be equally robust, because for both the maximal value of Γ for which they are feasible is 1. It can be shown that the path P_2 has a higher probability of staying below the delay limit than P_1 for many common probability distributions. This means that it would be desirable that P_2 is therefore deemed to be more robust.

As an example, let us consider the discrete uniform distribution for all edge delays. For P_1 we get 9 possible values from 4 to 12 for both delays. The path would exceed $B = 20$ in 4 cases if the first edge takes value 12. For values smaller than 12 there are 6 more cases. As a result we get a probability of

$$1 - \frac{4 + 3 + 2 + 1}{9 \cdot 9} = 88\%$$

that the delay of P_1 does not exceed B. But for P_2 there is the higher probability of

$$1 - \frac{1}{13 \cdot 3} = 97\%.$$

We would get similar results for many other probability distributions. So the solution consisting of P_2 has not the lowest weight, but the highest probability

$$Pr[\sum_{e \in P} \tilde{d}_e \leq B]$$

for all paths P to t, where $\tilde{d}_e$ is the random variable for the delay of edge e. It may be a loss that this solution to the instance is never classified as optimal for the RRDCST problem. Such undesirable effects are caused by the weak correlation of Γ and the quantiles of possible probability distributions. In Chapter 4 an approach is introduced which aims to evaluate paths more precisely.

3.2 Formulations

In this section several ILP formulations of the RRDCST problem are presented. Throughout the section the variable x_{ij} is used for an arc $(i, j) \in A$. If $x_{ij} = 1$ the arc (i, j) is part of the solution and if $x_{ij} = 0$ it is not. The formulations are directed, because directed formulations are in many cases stronger then undirected ones. Solutions will thus be represented as arborescences.

3.2.1 Multi Commodity Flow

The *multi commodity flow (MCF) formulation* is a compact formulation that uses only a polynomially bounded number of variables and constraints. A variable f_{ij}^k is used for the flow on arc $(i, j) \in A$ for terminal node $k \in T$.

To ensure connectivity, a flow from r to every terminal node is integrated into the formulation. First, we define a subproblem $\beta_k(\mathbf{f}^k, \Gamma)$, which is the part for handling the uncertain delays.

$$\beta_k(\mathbf{f}^k, \Gamma) = \max_{\substack{\{F \cup \{(u,v)\} | F \subseteq A, \\ |F| = \lfloor \Gamma \rfloor, (u,v) \in A \setminus F\}}} \{ \sum_{(i,j) \in F} \hat{d}_{ij} f_{ij}^k + (\Gamma - \lfloor \Gamma \rfloor) \hat{d}_{uv} f_{uv}^k \} \tag{3.4}$$

The set F is the set of arcs for which the highest delay variation is assumed. For a fractional part of Γ, also the delay of arc (v_1, v_2) is increased. The MCF formulation below is a non-linear program which includes the subproblem $\beta_k(\mathbf{f}^k, \Gamma)$.

$$\min \sum_{(i,j) \in A} w_{ij} x_{ij} \tag{3.5}$$

$$\text{s.t.} \sum_{(i,j) \in A} d_{ij} f_{ij}^k + \beta_k(\mathbf{f}^k, \Gamma) \leq B \qquad \forall k \in T \tag{3.6}$$

$$\sum_{(r,i) \in A} f_{ri}^k = 1 \qquad \forall k \in T \tag{3.7}$$

$$\sum_{(i,j) \in A} f_{ij}^k - \sum_{(j,i) \in A} f_{ji}^k = 0 \qquad \forall j \in V \setminus \{r, k\}, \forall k \in T \tag{3.8}$$

$$\sum_{(i,k) \in A} f_{ik}^k = 1 \qquad \forall k \in T \tag{3.9}$$

$$\sum_{(i,k) \in A} x_{ik} = 1 \qquad \forall k \in T \tag{3.10}$$

$$\sum_{(i,k) \in A} x_{ik} \leq 1 \qquad \forall k \in S \tag{3.11}$$

$$0 \leq f_{ij}^k \leq x_{ij} \qquad \forall (i,j) \in A, \forall k \in T \tag{3.12}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A \tag{3.13}$$

The inequalities (3.7), (3.8) and (3.9) are classical flow conservation constraints. Inequalities (3.12) links the flow variables to the arc variables. The delay constraints are described with inequalities (3.6). Differently as for the deterministic problem, there has to be taken more care that only tree like solutions are allowed. This can be done with constraints (3.10) and (3.11) which describe the in-degree of terminal and potential Steiner nodes. Alternatively, all flow variables could have been restricted explicitly to be integer to avoid partial flows on different paths. Next, we apply the transformation given in [6] to our model (3.5)–(3.13) to get an ILP formulation.

Let us consider $\beta_k(\mathbf{f}^k, \Gamma)$ again. Since $\mathbf{f}^k$ are constants for $\beta_k(\mathbf{f}^k, \Gamma)$ we can construct an equivalent linear program.

$$\beta_k(\mathbf{f}^k, \Gamma) = \max \sum_{(i,j)\in A} \hat{d}_{ij} f_{ij}^k z_{ij}^k \tag{3.14}$$

$$\text{s.t.} \sum_{(i,j)\in A} z_{ij}^k \leq \Gamma \qquad (q_k) \tag{3.15}$$

$$0 \leq z_{ij}^k \leq 1 \qquad (p_{ij}^k) \qquad \forall (i,j) \in A \tag{3.16}$$

In an optimal solution every newly introduced variable z_{ij}^k will be 1 if $(i,j) \in F$, between 0 and 1 if $(i,j) = (u,v)$ and 0 otherwise. Note that there is no need to handle the fractional part of Γ separately like it was done before.

Using strong duality we can construct an equivalent minimization problem. This is necessary, because $\beta_k(\mathbf{f}^k, \Gamma)$ appears on the left side of inequalities (3.6) which implies that the function is automatically minimized. The variables q_k correspond to inequality (3.15) and the variables p_{ij}^k correspond to the definition of the possible interval (3.16).

$$\min q_k \Gamma + \sum_{(i,j)\in A} p_{ij}^k \tag{3.17}$$

$$\text{s.t. } q_k + p_{ij}^k \geq \hat{d}_{ij} f_{ij}^k \qquad \forall (i,j) \in A \tag{3.18}$$

$$q_k \geq 0 \tag{3.19}$$

$$p_{ij}^k \geq 0 \qquad \forall (i,j) \in A \tag{3.20}$$

The final step is to integrate the subproblem again into the MCF formulation. So (3.6) has to be substituted with the constraints (3.21)–(3.24). There are no quadratic terms, although $\mathbf{f}^k$ cannot be viewed as constant like it was done before.

$$\sum_{(i,j)\in A} d_{ij} f_{ij}^k + q_k \Gamma + \sum_{(i,j)\in A} p_{ij}^k \leq B \qquad \forall k \in T \tag{3.21}$$

$$q_k + p_{ij}^k \geq \hat{d}_{ij} f_{ij}^k \qquad \forall (i,j) \in A, \forall k \in T \tag{3.22}$$

$$q_k \geq 0 \qquad \forall k \in T \tag{3.23}$$

$$p_{ij}^k \geq 0 \qquad \forall (i,j) \in A, \forall k \in T \tag{3.24}$$

3.2.2 Path-Cut

It is often the case that there is a more intuitive formulation of a problem where an exponential number of constraints is needed. The following formulation was already given by Ruthmair [44] and can be used to solve the RDCST problem as well as RRDCST problem. Note that contrary to the huge amount of constraints the number of different types of inequalities is very low. There are also no further variables needed than the arc variables $\mathbf{x}$.

Let $P = \{\{u_i, u_i + 1\} \mid i = 1, 2, \ldots, l-1\}$ be the edge set which corresponds to a node sequence $U = (u_i)_{i=1}^l$ that represents a directed path in graph G. Then the set $\mathcal{P}_{inf}$ contains a

node sequence U if $d_P^\Gamma > B$. Thus, $\mathcal{P}_{inf}$ contains only node sequences of paths that cannot be part of a feasible solution.

$$\min \sum_{(i,j)\in A} w_{ij}x_{ij} \tag{3.25}$$

$$\text{s.t.} \sum_{(i,j)\in A, i\in C, j\in V\setminus C} x_{ij} \geq 1 \qquad \forall C \subset V, r \in C, (V \setminus C) \cap T \neq \emptyset \tag{3.26}$$

$$\sum_{i=1}^{l-1} x_{u_i u_{i+1}} \leq l-2 \qquad \forall (u_i)_{i=1}^{l} \in \mathcal{P}_{inf} \tag{3.27}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.28}$$

The inequalities (3.26) are often called connection cut inequalities. They ensure that there is a connection between the root node and all terminal nodes. Constraints (3.27) forbid all infeasible paths. A separate method could define which paths are feasible and which are not. Therefore the formulation is flexible and able to describe different problems.

A common way to solve an ILP with an exponential number of constraints is branch-and-cut. Then constraints have to be added to the model only if needed. There are some other inequalities which are not necessary but could be inserted a priori to increase performance. Here, the constraints (3.10) and (3.11) can be used as such.

$$\sum_{(r,j)\in A} x_{rj} \geq 1 \tag{3.29}$$

$$\sum_{(i,k)\in A} x_{ik} \leq \sum_{(k,j)\in A} x_{kj} \qquad \forall k \in S \tag{3.30}$$

$$\sum_{(i,k)\in A, i\neq j} x_{ik} \geq x_{kj} \qquad \forall (k,j) \in A, k \in S \tag{3.31}$$

Inequality (3.29) is one special case of (3.26) and it is very likely that it would be added during the branch-and-cut procedure anyway. It can easily be seen that a potential Steiner node cannot be a leaf in an optimal solution which is formalized with constraints (3.30). Inequalities (3.31) also state that there can only be an outgoing arc from a potential Steiner node if there is an incoming arc.

There is also a stronger variant of constraints (3.27) which was given in [44]. Using the strengthened constraints (3.32) instead of (3.27) results in a much better performance of the branch-and-cut algorithm.

$$\sum_{i=1}^{l-1} x_{u_i u_{i+1}} + \sum_{i=2}^{l-1} \sum_{v\in V_U^i} x_{v u_i} + \sum_{i=2}^{l-1} x_{u_{i+1} u_i} \leq l-2 \qquad \forall (u_i)_{i=1}^{l} \in \mathcal{P}_{inf} \tag{3.32}$$

The node set $V_U^i = \{v \mid (v,u_i) \in A, v \neq u_{i-1}, v \neq u_{i+1}, (v,u_i,\ldots,u_l) \in \mathcal{P}_{inf}\}$ is used to forbid additional paths without additional inequalities.

3.2.2.1 Separation Methods

In the implementation there are two different separation methods, one for the connection cuts (3.26) and one for the infeasible paths (3.27) or (3.32). Sometimes it is beneficial if the LP solutions are strengthened at each node. Therefore both methods can identify violated inequalities given a fractional solution. For each LP or ILP solution, the search for violated constraints is done with both separation methods and not aborted when an inequality is added to the model. This could have been done also otherwise, and it is not easy to predict how many constraints should be added for one solution, in order to achieve the best performance with current ILP solvers for a specific set of instances. In the following, let $\mathbf{x}'$ be the vector of the current LP solution.

Connection Cuts The connection cut inequalities are checked with a maximum-flow algorithm [14]. For each terminal node t the maximum flow from r with arc capacities $\mathbf{x}'$ is calculated. If there is a path between r and t, the flow has to be at least one. If the flow is below one, at least one connection cut inequality is violated. The max-flow min-cut theorem says that the value of the flow is equal to the minimal cut. Some minimal cuts can also be found with the maximum-flow algorithm. One such cut corresponds to a violated inequality if it is below one. There is often a huge amount of them, therefore, to keep the model small, only the closest cut to source r and the closest cut to the terminal node are added.

Path Cuts The part where the uncertainty has to be handled is the second separation method. A directed path P on graph G is viewed as part of an LP solution if

$$\sum_{(i,j)\in P} x'_{ij} > |P| - 1, \tag{3.33}$$

which means that no whole arc can be missing. Such a path will violate an inequality (3.27) if $P \in \mathcal{P}_{inf}$.

First, such paths have to be located. This can be done with a shortest path algorithm where the weight of each arc (i, j) is set to $1 - x'_{ij}$. For each terminal node the shortest path from r is determined. If a shortest path is smaller than one, a feasibility check is done. Note that this way, no violated inequality will be found on an LP solution, if there is a shorter path for every terminal node which is not contained in the set $\mathcal{P}_{inf}$. This means that this method is a heuristic on LP solutions, but exact on ILP solutions.

The feasibility check of a path P from r to terminal node t works as follows. It is easy to verify if P violates a delay constraint (3.2). It is beneficial to search for shorter paths to reduce the number of path cuts. Therefore, if $P \in \mathcal{P}_{inf}$, the infeasible subpath with the least edges ending at t is used to construct a new path cut.

3.2.3 Layered Graph

The procedure which is used here transforms the instance graph to a much larger graph which is called layered graph and contains no cycles. Furthermore it contains no infeasible paths in the

case of certain delays and a relatively small number of infeasible paths in the case of uncertain delays. If the layered graph gets not too large, hop and delay-constrained tree problems can often be solved efficiently with this procedure. Therefore, layered graph transformations also gained popularity in recent work. Let us first consider only the RDCST problem, for which the following layered graph approach was presented in [44].

The layered graph has $B+1$ layers which will be called as layer 0 up to layer B. The number stands for the delay which it takes to get from r to one node in this layer. So layer 0 consists only of the root node itself. All other nodes have exactly one copy of themselves in each of the layers 1 to B. The nodes of the layered graph are defined by the set

$$V_L = \{r\} \cup \{v_b \mid v \in V \setminus \{r\}, 1 \leq b \leq B\}.$$

The number of nodes of the layered graph is then $B \cdot (|V| - 1) + 1$. Each arc $(u, v) \in A$ with delay d gets also several copies (u_b, v_{d+b}). More formally, the arc set of the layered graph is defined as $A_L = A_L^r \cup A_L^q$ where

$$A_L^r = \{(r, v_{d_{rv}}) \mid (r, v) \in A\} \text{ and}$$

$$A_L^q = \{(u_b, v_{b+d_{uv}}) \mid (u, v) \in A, u \neq r, 1 \leq b \leq B - d_{uv}\}.$$

A layered graph transformation for the hop-constrained spanning tree problem was introduced by Gouveia et al. [26]. They added also further arcs with weight 0 from each node to the copy of the node on the highest layer. This way the problem can be solved as a Steiner arborescence problem on the layered graph. To show the validity of the transformation, it was discussed why each feasible solution on the layered graph has an equivalent counterpart on the original graph and the other way round. If these additional arcs are added to the set A_L, also the RDCST problem can be solved as a Steiner arborescence problem.

A crucial factor for the efficiency of the optimization process is the size of the layered graph. The number of arcs can be reduced by recursively applying the following rules:

- If a node has no incoming arcs, all outgoing arcs can be deleted.
- If a potential Steiner node has no outgoing arcs, all incoming arcs can be deleted.

For an arc (i_b, j_c) of the layered graph the variable x_{ij}^b is introduced. An arc is part of the solution if $x_{ij}^b = 1$, and it is not if $x_{ij}^b = 0$. Then the deterministic problem can be formulated as follows:

$$\min \sum_{(i,j)\in A} w_{ij}x_{ij} \tag{3.34}$$

$$\text{s.t.} \sum_{(i_b,k_c)\in A_L} x^b_{ik} = 1 \qquad \forall k \in T \tag{3.35}$$

$$\sum_{(i_b,k_c)\in A_L, i\neq j} x^b_{ik} \geq x^c_{kj} \qquad \forall (k_c, j_d) \in A^g_L \tag{3.36}$$

$$\sum_{(i_b,j_c)\in A_L} x^b_{ij} = x_{ij} \qquad \forall (i,j) \in A \tag{3.37}$$

$$x^b_{ij} \geq 0 \qquad \forall (i_b, j_c) \in A_L \tag{3.38}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.39}$$

The inequalities (3.35) allow exactly one incoming arc for all copies of one terminal node. For every outgoing arc of a node different to r, there has to be a supporting incoming arc, which is described with (3.36). Since the original arc variables are used in the objective function, the linking constraints (3.37) are necessary. This way we get not only the solution on the layered graph, but also the solution for the original problem immediately.

3.2.3.1 Connection Cuts on the Layered Graph

In order to strengthen the formulation connection cuts can be added. The constraints (3.26) on the original graph are of course valid inequalities. But there is also an even larger amount of connection cuts on the layered graph. The corresponding inequalities were introduced in [26] and can be described with constraints (3.40).

$$\sum_{(i_b,j_c)\in A_L, i_b\in C, j_c\in V_L\setminus C} x^b_{ij} \geq 1 \qquad \begin{array}{c} \forall C \subset V_L, r \in C, \\ C \cap \{k_b \mid 1 \leq b \leq B\} = \emptyset, k \in T \end{array} \tag{3.40}$$

3.2.3.2 Working with Uncertain Delays

Unfortunately, it is not so easy to construct a layered graph without infeasible paths if delays are uncertain. One possibility to extend the layered graph approach to the RRDCST problem is to add the path constraints (3.27) or (3.32) to the model. Like in Section 3.2.2, they are best added dynamically in a cutting plane algorithm. The layered graph is built according to the expected edge delays.

As defined in Section 3.2, it has to hold for a feasible path P that

$$d_P = \sum_{e\in P} d_e \leq B - \hat{d}^{\Gamma}_P.$$

We are now looking for an upper bound for d_P for an arbitrary path. For every edge e, there is a β_e, $0 < \beta_e < 1$, with $\hat{d}_e = \beta_e \cdot d_e$. We define $\beta_{\min}$ as the minimal factor for all edges.

The edge set E_s is used to calculate a lower bound for the delay variation of a path with a high total delay and constructed the following way:

1. E_s is empty.
2. The edge e with the smallest delay variation $\hat{d}_e$ is taken from $E \setminus E_s$. Its expected delay is increased to $\frac{\hat{d}_e}{\beta_{\min}}$.
3. If $d^{\Gamma}_{E_s \cup \{e\}} \leq B$, e is added to E_s and it is continued with 2.

Every feasible path contains edges which cannot have smaller delay variations than those encountered in E_s, because they are already the smallest. There are feasible paths with less edges, but these cannot have high expected delays. This follows from the calculation with $\beta_{\min}$ which creates an upper bound on d_P for every $\hat{d}^{\Gamma}_P$. Therefore, there is no feasible path P with $d_P > B - \hat{d}^{\Gamma}_{E_s}$.

We get an upper bound

$$L = \lfloor B - \hat{d}^{\Gamma}_{E_s} \rfloor \tag{3.41}$$

for the last layer in the layered graph that has to be considered. Note that $L < B$. This means that B is also an upper bound, but the smaller bound L reduces the layered graph and increases the performance of the optimization process.

Theorem 3.1 *L is an upper bound for d_P of every feasible path P of a given instance of the RRDCST problem.*

Proof. Let us assume to the contrary that a feasible path P with $d_P > L$ exists. Since d_P has to be integer, it has to hold that $d_P > B - \hat{d}^{\Gamma}_{E_s}$. It follows directly that $\hat{d}^{\Gamma}_P < \hat{d}^{\Gamma}_{E_s}$.

We distinguish two cases:

1. P contains less than $\lceil \Gamma \rceil$ edges. Then, an upper bound on d_P follows directly from the definition of d^{Γ}_P. Together with $\hat{d}^{\Gamma}_P < \hat{d}^{\Gamma}_{E_s}$ we get a contradiction.

$$d_P \leq \sum_{e \in P} \frac{\hat{d}_e}{\beta_{\min}} \leq B - \hat{d}^{\Gamma}_{E_s}$$

2. P contains at least $\lceil \Gamma \rceil$ edges. This contradicts with $\hat{d}^{\Gamma}_P < \hat{d}^{\Gamma}_{E_s}$, because $\hat{d}^{\Gamma}_{E_s}$ contains the $\lceil \Gamma \rceil$ lowest delay variations.

So there is no feasible path P with $d_P > L$. □

As an example, let us consider the graph given in Fig. 3.4. The calculation of the number of layers does not respect the structure of the graph, but it uses the edge delays and delay variations. Table 3.1 summarises this information. Additionally, the factor $\beta_e = \hat{d}_e / d_e$ is calculated for every edge e.

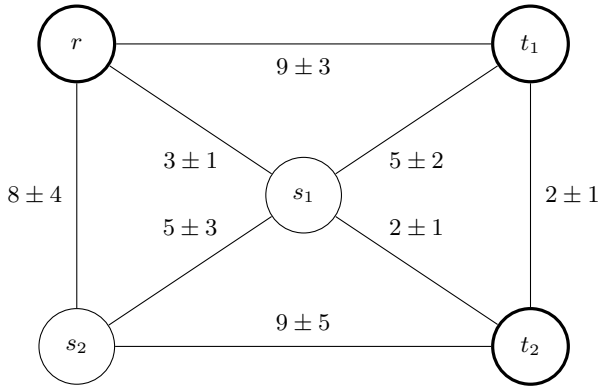


Figure 3.4: Graph and edge delays of an instance of the RRDCST problem. We have $S = \{s_1, s_2\}$ and $T = \{t_1, t_2\}$. Each edge e is labelled with $d_e \pm \hat{d}_e$.

Table 3.1: Edge delays and delay variations of Fig. 3.4.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
$\hat{d}$	1	1	1	2	3	3	4	5
d	2	2	3	5	5	9	8	9
β	0.5	0.5	0.33	0.4	0.6	0.33	0.5	0.56

Table 3.2: Increased edge delays.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
$\hat{d}$	1	1	1	2	3	3	4	5
d	3	3	3	6	9	9	12	15

We get $\beta_{\min} = \frac{1}{3}$. All delays are increased to $\hat{d}_e \cdot 3$ for the following calculation steps. So we work with the values given in Table 3.2.

Let us assume that the instance also specifies that $B = 20$ and $\Gamma = 2$. In the first three iterations the edges e_1, e_2 and e_3 are added to the set E_s without exceeding the delay bound. The next edge is e_4. We get

$$d^{\Gamma}_{E_s \cup \{e_4\}} = 6 + 3 + 3 + 3 + 2 + 1 = 18.$$

The calculated value stays below the bound, therefore e_4 is added to E_s. For e_5 the delay of the edge set

$$d^{\Gamma}_{E_s \cup \{e_5\}} = 9 + 6 + 3 + 3 + 3 + 3 + 2 = 29 > B.$$

The edge e_5 is not added to E_s. As a result, the number of layers can be set to

$$B - \hat{d}^{\Gamma}_{E_s} = 20 - 2 - 1 = 17.$$

3.2.4 Path

Also ILP formulations with polynomially many constraints but exponentially many variables are often used. These problems are then usually solved with a branch-and-price algorithm. Such a formulation was introduced by Gouveia et al. [25] for the spanning tree variant of the RDCST problem. This could be seen as a special case with $S = \emptyset$. An adaption to the RDCST problem was already given by Leitner et al. [36]. The formulation can be used without any change when working with uncertain delays. The only part that is different for the RRDCST problem is the set of variables which affects the pricing subproblem.

Here we are looking for the set of feasible paths in contrast to the set of infeasible paths in Section 3.2.2. It is sufficient to consider only feasible directed paths that go from the root node to a terminal node. For every node $t \in T$ we define the set $\mathcal{P}_t$ which contains the arc set of every path P from r to t with $d_P^\Gamma \leq B$. The whole set of paths is defined as $\mathcal{P} = \bigcup_{t \in T} \mathcal{P}_t$. For each path $P \in \mathcal{P}$ there is a variable λ_P which is 1 if P is part of the solution and 0 otherwise. The formulation is given the following way:

$$\min \sum_{(i,j) \in A} w_{ij} x_{ij} \tag{3.42}$$

$$\text{s.t.} \sum_{P \in \mathcal{P}_k} \lambda_P = 1 \qquad \forall k \in T \tag{3.43}$$

$$\sum_{P \in \mathcal{P}_k, (i,j) \in P} \lambda_P \leq x_{ij} \qquad \forall (i,j) \in A, \forall k \in T \tag{3.44}$$

$$\sum_{(i,v) \in A} x_{iv} \leq 1 \qquad \forall v \in V \setminus \{r\} \tag{3.45}$$

$$\lambda_P \geq 0 \qquad \forall P \in \mathcal{P} \tag{3.46}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.47}$$

The inequalities (3.43) ensure that there is one path from r to every terminal node. There are also linking constraints between arc and path variables needed which can be described with (3.44). The constraints (3.45) were similarly used before and limit the number of incoming arcs for every node.

3.2.4.1 Pricing Subproblem

As described in [25] the pricing problem can be stated as follows:

Let us define γ_{ij}^k as the non-negative dual variables of constraints (3.44) in the form

$$x_{ij} - \sum_{P \in \mathcal{P}_k, (i,j) \in P} \lambda_P \geq 0$$

and β^k as the dual variables of constraints (3.43). If there is a path $P \in \mathcal{P}_k$ for some $k \in T$ in

the instance graph with

$$\sum_{(i,j)\in P} \gamma_{ij}^k < \beta^k \tag{3.48}$$

and a satisfied delay constraint, then there is a corresponding path variable with negative reduced cost which should be added to the restricted model. Note that this is a feasibility problem which is an optimization problem without an objective function. Although the solution with the minimum reduced cost may be the most promising candidate for a new path variable. For the RDCST problem this can be determined with a constrained shortest path problem for each terminal node. This problem can be solved in polynomial time with respect to the size of the instance and B.

For the RRDCST problem it might be beneficial to minimize the delay instead. This results in a robust constrained shortest path problem for each terminal node k where the weights and the delays are swapped. The weight of an edge e is then defined with expectation d_e and its maximal variation $\hat{d}_e$. The delay of an arc (i, j) is then given by γ_{ij}^k and constraint (3.48) has to hold. This problem still can be solved in polynomial time by solving multiple deterministic problems. This can be concluded from [5], where it was proven that a reduction to a deterministic counterpart is possible for a large class of robust problems.

3.2.5 Miller-Tucker-Zemlin

The idea of the formulation of Miller, Tucker and Zemlin [39] is to assign a number to every node. This way some constraints can easily be checked. One example is the RDCST problem, see Fig. 3.5a, where both the delay constraints and the connectedness can be verified together. But the concept relies on the fact that it is possible to determine the number from local information which are incident edges and adjacent nodes. Figure 3.5b illustrates, that global information is necessary in the RRDCST problem. For the first node $d_e + \hat{d}_e = 4 + 2 = 6$ is counted for the incoming edge e. For the left node only the expected delay of the incident edge is added because of $\Gamma = 1$. For the right node it is not possible to reach 13 as a result only with local information.

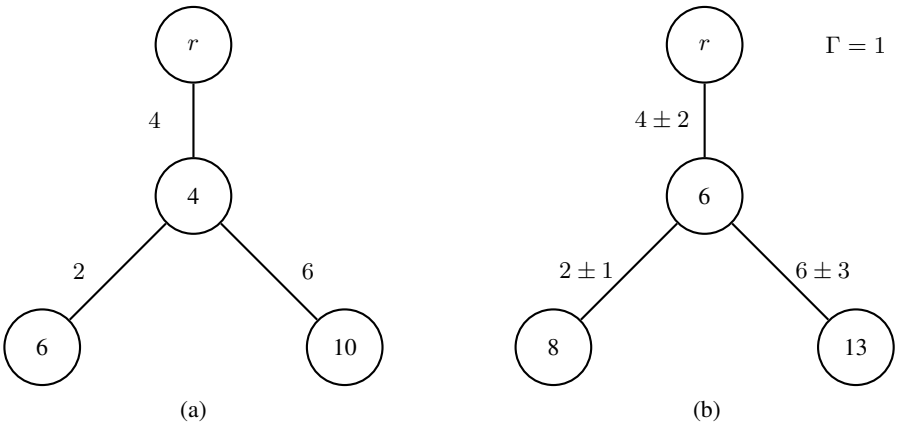


Figure 3.5: (a) A solution of a deterministic problem instance and (b) a solution of an instance of the RRDCST problem. Each edge e is labelled with d_e and $d_e \pm \hat{d}_e$, respectively. The calculated delay, d_P for (a) and d_P^Γ for (b), of the path P from r is written to each node.

CHAPTER 4

The Stochastic Rooted Delay-Constrained Steiner Tree Problem

The expected case of the RRDCST problem is equivalent to the deterministic problem, and as we will see, the same holds for the stochastic problem variant defined in Section 4.2. For bad cases we have not yet defined how bad they exactly are, because it was not possible with the robust approach. Let us assume we are given a probability p and want to find solutions where each path is feasible with probability $\geq p$. In this chapter p characterises the case for which the optimal solution should be found. Good cases are equal to $0 < p < 0.5$, the expected case to $p = 0.5$ and good cases to $0.5 < p < 1$. The probability $p = 1$ would describe the worst case, but it will be assumed that there are no worst case solutions for instances of the stochastic problem.

4.1 Normally Distributed Approach

This section analyses chance constraints which are not problem specific, but based on a concrete model with the assumptions:

1. The random variables $\tilde{\mathbf{a}}$ are independent.
2. Every $\tilde{a}_j$ is normally distributed.

The first assumption of the model is crucial as it was for the interval-based model. The second assumption allows precise calculations, but if it is violated, it should still lead to good approximations of the quantiles.

Let us first consider the random variable X which is the sum of random variables $\tilde{\mathbf{a}}$.

$$X = \sum_{j \in J} \tilde{a}_j$$

The goal is to find the p-quantile of X which is denoted as $Q_X(p)$. The deterministic constraints (4.1) can then be substituted by inequalities (4.2).

$$\sum_{j \in J} a_j \leq b \tag{4.1}$$

$$Q_X(p) \leq b \tag{4.2}$$

The central limit theorem [52] states that the sum of a large number of independent random variables is approximately normally distributed. This means that for a large set J the quantile can be calculated approximately with (4.3) where the expected value of a random variable X is denoted with $E[X]$ and its standard deviation with $\mathrm{stdev}(X)$. The quantile of the normal distribution with zero as its mean and with variance equal to one is denoted with u_p. For a given p the quantile u_p is constant and because of the symmetry of the normal distribution it holds that $u_{0.5} = 0$. Furthermore, it follows that $u_p < 0$ for good cases and $u_p > 0$ for bad cases. Because of the independence of the random variables $\tilde{\mathbf{a}}$, (4.3) is equal to (4.4).

$$Q_X(p) \approx E[\sum_{j \in J} \tilde{a}_j] + u_p \cdot \mathrm{stdev}(\sum_{j \in J} \tilde{a}_j) \tag{4.3}$$

$$= \sum_{j \in J} E[\tilde{a}_j] + u_p \cdot \sqrt{\sum_{j \in J} \mathrm{stdev}(\tilde{a}_j)^2} \tag{4.4}$$

This allows us to work precisely with approximately normally distributed values which we naturally get for large sums of random variables. If all random variables are normally distributed from the beginning, this precision is extended to all inequalities. And even if they are not, the error may be very small. Besides this, many random variables are indeed approximately normally distributed in practice.

The model which is used here is different to the one that was introduced for the robust approach. In theory there is no need that models in robust optimization must be different to models in stochastic programming. The difference is caused by computational aspects. It would be much more complicated to calculate quantiles if the model included worst cases. The differences in the model make it a bit harder to compare the robust and the stochastic approach. On the other hand there is the advantage that there is more flexibility for a real world problem. It could be determined if there are worst cases, and the approach can be chosen accordingly.

There was a similar approach introduced by Ben-Tal and Nemirovski [4] but in a robust fashion. Instead of the standard deviation they used worst case values and basically the same assumptions as in Section 2.2.1. The worst case value is taken for a variable if the quantile of the normal distribution would be even higher. Because of this, the approach cannot be categorized to stochastic programming anymore. No matter what assumptions are made for the probability distribution, the stochastic precision is lost. An exception are bad cases near the expectation. For those the approach of Ben-Tal and Nemirovski is equivalent to the one described in this chapter. Their approach can be modelled as a *second order cone program*. Though this class of problems is often described as tractable, second order cone programs are not as attractive as linear programs from a computational point of view. This holds especially for their integer and mixed integer counterparts.

4.2 The Stochastic Problem

The stochastic problem that follows the normally distributed approach is defined as follows:

Definition 4.1 (Stochastic Rooted Delay-Constrained Steiner Tree (SRDCST) Problem) *We are given a graph $G = (V, E)$, a delay bound $B \in \mathbb{N}$ and a quantile $u_p \in \mathbb{Q}$. The set of vertices V is partitioned into a root node r, a set of terminal nodes $T \subseteq V \setminus \{r\}$ and a set of potential Steiner nodes $S = V \setminus (T \cup \{r\})$. Each edge $e \in E$ has an assigned weight $w_e \in \mathbb{N}$, an expected delay $d_e \in \mathbb{N}$ and a standard deviation $\check{d}_e \in \mathbb{Q}$ with $0 \leq \check{d}_e < d_e$.*

A feasible solution is a tree $G' = (V', E')$ with $T \cup \{r\} \subseteq V' \subseteq V$ and $E' \subseteq E$, fulfilling the delay constraints

$$d^p_{P_{G'}(r,t)} = \sum_{e \in P_{G'}(r,t)} d_e + u_p \cdot \sqrt{\sum_{e \in P_{G'}(r,t)} \check{d}_e^2} \leq B, \forall t \in T, \tag{4.5}$$

where $P_{G'}(r,t)$ is the edge set of the unique path from r to node t in tree G'. As it was described in Section 4.1, $d^p_{P_{G'}(r,t)}$ is the p-quantile of this path. The task is to find a feasible solution with minimal weight

$$w_{G'} = \sum_{e \in E'} w_e. \tag{4.6}$$

Definition 4.2 *The* α-SRDCST problem *is a special case of the SRDCST problem, where a constant $\alpha \in \mathbb{Q}$, $0 < \alpha < 1$, exists such that $\check{d}_e = \alpha \cdot d_e, \forall e \in E$.*

Figure 3.1 does not only fit to the 0.5-RRDCST problem, but it shows also an instance of the 0.5-SRDCST and an optimal solution to it with $u_p = 1$. The calculated delays of the three paths are then slightly different. For a path P_i to t_i we get the following values:

$$d^p_{P_1} = 11 + \sqrt{2.5^2 + 2^2 + 1^2} = 14.35$$

$$d^p_{P_2} = 11 + \sqrt{4.5^2 + 1^2} = 15.61$$

$$d^p_{P_3} = 16 + \sqrt{2.5^2 + 2^2 + 1^2 + 1.5^2 + 1^2} = 19.81$$

It is not a rare case that the RRDCST and the SRDCST problem have the same optimal solutions for an instance with $\check{d}_e = \hat{d}_e, \forall e \in E$, a fixed value of Γ and some $u_p \leq \Gamma$. But that does not mean that the two problems are interchangeable which will be demonstrated in Section 7.3.

4.3 Formulations

This section is about ILP formulations of the SRDCST problem. Surprisingly, the differences in the formulations between the robust and the stochastic RDCST are except for Section 4.3.1 only marginal.

4.3.1 Multi Commodity Flow

For the SRDCST problem a MCF formulation with quadratic constraints can be given with (4.7), (4.8) and the previously introduced constraints (3.7)–(3.13).

$$\min \sum_{(i,j)\in A} w_{ij}x_{ij} \tag{4.7}$$

$$\text{s.t.} \sum_{(i,j)\in A} d_{ij}f_{ij}^k + u_p \cdot \sqrt{\sum_{(i,j)\in A} \check{d}_{ij}^2 f_{ij}^k} \leq B \qquad \forall k \in T \tag{4.8}$$

However, such a formulation is clearly not as attractive as the MCF formulation from Section 3.2.1, because current solvers can't solve it as efficiently as a linear program.

4.3.2 Path-Cut

For the SRDCST problem there are only small differences to Section 3.2.2. The set $\mathcal{P}_{inf}$ contains a node sequence if $d_P^p > B$ for the corresponding edge set P. This affects only the path cuts of the branch-and-cut algorithm. The feasibility check has to verify if a path violates a delay constraint (4.5). This has no further effect on the separation method for both LP and ILP results.

4.3.3 Layered Graph

As in the path-cut formulation $\mathcal{P}_{inf}$ describes a different set as in Section 3.2.3. This affects not only the path cuts, but also the number of layers.

For the SRDCST problem it may not be enough to stop at layer B, because the parameter u_p is allowed to take negative values. It is defined in Section 4.1 that for a feasible path P it has to hold that

$$d_P = \sum_{e\in P} d_e \leq B - u_p \cdot \sqrt{\sum_{e\in P} \check{d}_e^2}.$$

We are now looking for an upper bound for d_P for an arbitrary path. For every edge e there is a β_e, $0 < \beta_e < 1$, with $\check{d}_e = \beta_e \cdot d_e$. We define $\beta_{\min}$ as the minimal and $\beta_{\max}$ as the maximal factor for all edges.

The edge sets E_h and E_s are used to calculate a lower bound for the delay variation of a path with a high total delay. The set E_h is used for good cases and constructed the following way

1. E_h is empty.
2. The edge e with the highest standard deviation $\check{d}_e$ is taken from $E \setminus E_h$. Its expected delay is decreased to $\frac{\check{d}_e}{\beta_{\max}}$ and it is added to E_h.
3. If $d_{E_h}^p < B$, it is continued with 2.

The set E_s is used for bad cases and constructed with the following steps:

1. E_s is empty.

2. The edge e with the smallest standard deviation $\check{d}_e$ is taken from $E \setminus E_s$. Its expected delay is increased to $\frac{\check{d}_e}{\beta_{\min}}$.

3. If $d^p_{E_s \cup \{e\}} \leq B$, e is added to E_s and it is continued with 2.

Then we distinguish two cases. Like for the RRDCST problem the two sets can be used to describe the upper bounds. For $u_p < 0$ we get

$$L_1 = \lfloor B - u_p \cdot \sqrt{\sum_{e \in E_h} \check{d}_e^2} \rfloor \tag{4.9}$$

for the last layer in the layered graph that has to be considered. Similarly we get the bound

$$L_2 = \lfloor B - u_p \cdot \sqrt{\sum_{e \in E_s} \check{d}_e^2} \rfloor \tag{4.10}$$

for the case $u_p \geq 0$. Note that $L_1 \geq B$ and $L_2 < B$. Taking L_2 instead of B in the latter case is not necessary but increases the performance.

Theorem 4.1 *For a given instance of the SRDCST problem we have:*

1. *L_1 is an upper bound for d_P of every feasible path P if $u_p < 0$*
2. *L_2 is an upper bound for d_P of every feasible path P if $u_p \geq 0$*

Due to the comparison with the robust approach this thesis focuses more on bad cases and therefore we will prove only the second part of Theorem 4.1.

Proof. Let us assume to the contrary that a feasible path P with $d_P > L_2$ exists. Since d_P has to be integer, it has to hold that $d_P > B - u_p \cdot \sqrt{\sum_{e \in E_s} \check{d}_e^2}$. It follows directly that $\sum_{e \in P} \check{d}_e^2 < \sum_{e \in E_s} \check{d}_e^2$.

Let f be the edge with the highest standard deviation in E_s. We can subtract all common terms from both sum of squares which excludes the edges from $P \cap E_s$. This leaves only edges with a standard deviation of at least $\check{d}_f$ in the sum of squares of P.

$$\sum_{e \in P} \check{d}_e^2 < \sum_{e \in E_s} \check{d}_e^2 \qquad \Rightarrow \qquad \sum_{e \in P, \check{d}_e \geq \check{d}_f} \check{d}_e^2 < \sum_{e \in E_s \setminus P} \check{d}_e^2$$

If the sum of squares of the larger values is smaller, then this has also to hold for the sum of the values.

$$\sum_{e \in P, \check{d}_e \geq \check{d}_f} \check{d}_e < \sum_{e \in E_s \setminus P} \check{d}_e$$

Then, an upper bound of d_P can be determined which contradicts the assumption.

$$d_P \leq \sum_{e \in P} \frac{\check{d}_e}{\beta_{\min}} < \sum_{e \in E_s} \frac{\check{d}_e}{\beta_{\min}} \leq B - u_p \cdot \sqrt{\sum_{e \in E_s} \check{d}_e^2}$$

So there is no feasible path P with $d_P > L_2$. □

CHAPTER 5

Preprocessing

In complexity theory the runtime of an algorithm is always determined with respect to the size of the instance. The size is usually defined as the length in bits which is needed to encode the instance. More fine-grained runtime analyses on graphs often define the size of a graph as the number of its nodes or edges. This is mainly done if it is a goal to determine differences between sparse and dense graphs in the runtime complexity. We know that the higher the complexity, the higher the difference in the runtime for two similar instance sizes. This already motivates a preprocessing step. If it is possible to reduce the number of edges of the instance without a high computational effort, the worst-case runtime will be reduced even to a higher extent which should also decrease the actual runtime. But it should also be noted that the actual runtime of a program also depends on the structure of the whole instance. A good example for this dependence is the quicksort algorithm [28]. There are two different terms for the complexity, one for the average and one for the worst case. The actual runtime then depends not only on the instance size, but also on the value of the pivot elements. This means that if an element of the list is deleted, the choice of the pivot elements may be worse for the shorter list. The impact on the runtime can be hard to predict, though a drastic increase is rather unlikely. Similarly it could be argued for the RDCST problem that a preprocessing may only delete edges which would have a very small impact on the actual runtime of a program. So it can be said that a fast preprocessing should either decrease the overall runtime, maybe even drastically, or have no great impact on it.

It depends largely on the instance how many parts of the graph are not needed to find at least one optimal solution. Especially properties like density of the graph, the number of valid triangle inequalities and the delay bound have a great influence. The preprocessing for the RDCST problem was already discussed in [44].

5.1 Comparison with Paths

If there is an edge $\{u, v\}$ and a path $P(u, v) \subseteq E \setminus \{\{u, v\}\}$ with smaller or equal weight and for all cases also smaller or equal delay, the edge can be safely deleted. Then there is no reason

to prefer the edge over the path, and thus it will be still possible to find an optimal solution. The rule will be applicable more often if the triangle inequality does not hold for the weights and edge delays of the instance. But even otherwise it may be possible to delete edges with this rule. Let us consider the simpler case where the triangle inequality holds only for the weights. This means that $w_{uv} \leq w_{P(u,v)}$, so it is still possible that both are equal.

$$w_{uv} \geq \sum_{e \in P(u,v)} w_e \qquad \wedge \qquad d_{uv} \geq \sum_{e \in P(u,v)} d_e \tag{5.1}$$

For the deterministic problem there is only condition (5.1) to verify before deleting an edge. This preprocessing step is more difficult for the RRDCST and the SRDCST problem and there are again some differences between them.

RRDCST Let us assume that there is a path P in a solution of the RRDCST problem. This path contains a subpath P_{ru} from r to u and a subpath P_{vt} from v to a terminal node t. The remaining part from u to v either consists of the edge $\{u, v\}$ or the path $P(u, v)$. Let us further assume that for both possibilities the highest $\lceil \Gamma \rceil$ values of the delay variations lie in $P_{ru} \cup P_{vt}$. Then the part between u and v would not influence the additional delay $\hat{d}^{\Gamma}_{P}$. This gives us the result that we still have to check the expected delays with condition (5.1).

Let us consider another scenario next where the path P only consists of either $\{u, v\}$ or $P(u, v)$, because $u = r$ and v is a terminal node. This means that condition (5.2) would have to be checked.

$$d_{uv} + \min(1, \Gamma) \cdot \hat{d}_{uv} \geq \sum_{e \in P(u,v)} d_e + \hat{d}^{\Gamma}_{P(u,v)} \tag{5.2}$$

These are only two possible scenarios, but it is sufficient to check only conditions (5.1) and (5.2).

Theorem 5.1 *An edge $\{u, v\}$ cannot be part of every optimal solution of a given instance of the RRDCST problem if* (5.1) *and* (5.2) *holds.*

Proof. Let us consider an arbitrary path P_2 from r to a terminal node t which contains $P(u, v)$. We define the edge set $F = P_2 \setminus P(u, v)$ and path $P_1 = F \cup \{\{u, v\}\}$. We have

$$d^{\Gamma}_{P_1} = \sum_{e \in F} d_e + d_{uv} + \hat{d}^{\Gamma}_{P_1} \text{ and}$$

$$d^{\Gamma}_{P_2} = \sum_{e \in F} d_e + \sum_{e \in P(u,v)} d_e + \hat{d}^{\Gamma}_{P_2}.$$

We assume that conditions (5.1) and (5.2) hold. We have to show that $d^{\Gamma}_{P_1} \geq d^{\Gamma}_{P_2}$ follows. By subtracting the common term we get

$$d_{uv} + \hat{d}^{\Gamma}_{P_1} \geq \sum_{e \in P(u,v)} d_e + \hat{d}^{\Gamma}_{P_2}. \tag{5.3}$$

We distinguish three cases:

1. $\hat{d}_{uv}$ influences $\hat{d}^{\Gamma}_{P_1}$ and $\Gamma \leq 1$. This implies that $\hat{d}_{uv}$ is bigger than $\hat{d}_e$, $\forall e \in F$. Because of $\Gamma \leq 1$ only a delay variation of one edge e can influence $\hat{d}^{\Gamma}_{P_2}$. We distinguish:
   - $e \in P(u,v)$. Then (5.2) directly implies (5.3).
   - $e \in F$. Then (5.1) directly implies (5.3).
2. $\hat{d}_{uv}$ influences $\hat{d}^{\Gamma}_{P_1}$ and $\Gamma > 1$. If there is no delay variation $\hat{d}_e$, $e \in F$, that influences only $\hat{d}^{\Gamma}_{P_2}$, but not $\hat{d}^{\Gamma}_{P_1}$, (5.2) implies (5.3). So we assume that such a delay variation $\hat{d}_e$ exists. Since the highest $\lceil \Gamma \rceil - 1$ delay variations from F influence $\hat{d}^{\Gamma}_{P_1}$, there can be at most one such edge. As a consequence, no delay variation from $P(u,v)$ can influence $\hat{d}^{\Gamma}_{P_2}$. But $\hat{d}_e$ has to be less than or equal to $\hat{d}_{uv}$. So (5.1) implies (5.3).
3. $\hat{d}_{uv}$ has no influence on $\hat{d}^{\Gamma}_{P_1}$. If there is no delay variation $\hat{d}_e$, $e \in F$, that influences only $\hat{d}^{\Gamma}_{P_1}$, but not $\hat{d}^{\Gamma}_{P_2}$, (5.1) implies (5.3). So we assume that such a delay variation $\hat{d}_e$ exists. But $\hat{d}_e$ has to be greater than or equal to $\hat{d}_{uv}$. So (5.2) implies (5.3).

Thus, (5.1) $\wedge$ (5.2) implies (5.3). □

It can be observed, that for the α-RRDCST problem condition (5.1) implies condition (5.2). This can be shown the following way:

$$d_{uv} \geq \sum_{e \in P(u,v)} d_e \quad \Rightarrow \quad \hat{d}_{uv} \geq \sum_{e \in P(u,v)} \hat{d}_e \quad \Rightarrow$$

$$\min(1, \Gamma) \cdot \hat{d}_{uv} \geq \hat{d}^{\Gamma}_{P(u,v)}$$

The first implication follows directly from the definition of the factor α. For the second implication there are two different cases. For $\Gamma < 1$ there can only be one edge where the delay variation is counted. But the delay variation of one edge in the path cannot be larger than $\hat{d}_{uv}$. For $\Gamma \geq 1$ we would already consider the worst case for the path only consisting of edge $\{u, v\}$. This cannot be smaller than the worst case for $P(u,v)$ which corresponds to $\Gamma \geq |P(u,v)|$. Thus only (5.1) has to be checked for the α-RRDCST problem.

SRDCST We will again consider the two extreme cases but now for the SRDCST problem. Let us assume that there occurs a path P in a solution. This path contains a subpath P_{ru} from r to u and a subpath P_{vt} from v to a terminal node t. The remaining part from u to v either consists of the edge $\{u, v\}$ or the path $P(u,v)$. Let us further assume that there is an edge in $P_{ru} \cup P_{vt}$ with a much higher standard deviation than all edges in $\{u, v\} \cup P(u,v)$. The standard

deviations of the part between u and v could have an arbitrarily small impact on d_P^p. This gives us the result that we still have to check the expected delays with condition (5.1).

In the other scenario the path P only consists of either $\{u, v\}$ or $P(u, v)$, because $u = r$ and v is a terminal node. This means that condition (5.4) would have to be checked.

$$d_{uv} + u_p \cdot \check{d}_{uv} \geq d_{P(u,v)} + u_p \cdot \sqrt{\sum_{e \in P(u,v)} \check{d}_e^2} \tag{5.4}$$

Similarly as for the RRDCST problem it can be shown that it is sufficient to check these two conditions.

Theorem 5.2 *An edge $\{u, v\}$ cannot be part of every optimal solution of a given instance of the SRDCST problem if* (5.1) *and* (5.4) *holds.*

Proof. Let us consider an arbitrary path P_2 from r to a terminal node t which contains $P(u, v)$. We define the edge set $F = P_2 \setminus P(u, v)$ and path $P_1 = F \cup \{\{u, v\}\}$. We have

$$d_{P_1}^p = d_F + d_{uv} + u_p \cdot \sqrt{\sum_{e \in P_1} \check{d}_e^2} \text{ and}$$

$$d_{P_2}^p = d_F + d_{P(u,v)} + u_p \cdot \sqrt{\sum_{e \in P_2} \check{d}_e^2}.$$

We assume that conditions (5.1) and (5.4) hold. We have to show that $d_{P_1}^p \geq d_{P_2}^p$ follows. By subtracting the common term we get (5.5).

$$d_{uv} + u_p \cdot \sqrt{\sum_{e \in P_1} \check{d}_e^2} \geq d_{P(u,v)} + u_p \cdot \sqrt{\sum_{e \in P_2} \check{d}_e^2}. \tag{5.5}$$

This can also we written as

$$d_{uv} + u_p \cdot \sqrt{\check{d}_{uv}^2 + \sum_{e \in F} \check{d}_e^2} \geq d_{P(u,v)} + u_p \cdot \sqrt{\sum_{e \in P(u,v)} \check{d}_e^2 + \sum_{e \in F} \check{d}_e^2}.$$

Let $c = d_{uv} - d_{P(u,v)}$ and $c' = \sum_{e \in F} \check{d}_e^2$.

$$c + u_p \cdot \sqrt{\check{d}_{uv}^2 + c'} \geq u_p \cdot \sqrt{\sum_{e \in P(u,v)} \check{d}_e^2 + c'} \tag{5.6}$$

Similarly, (5.4) results in (5.7).

$$c + u_p \cdot \check{d}_{uv} \geq u_p \cdot \sqrt{\sum_{e \in P(u,v)} \check{d}_e^2} \tag{5.7}$$

Since $c' \geq 0$, the difference of the standard deviations can only decrease for the longer paths.

$$\left| \check{d}_{rv} - \sqrt{\sum_{e \in P_{rv}} \check{d}_e^2} \right| \geq \left| \sqrt{\check{d}_{rv}^2 + c'} - \sqrt{\sum_{e \in P_{rv}} \check{d}_e^2 + c'} \right|$$

But this means that inequality (5.7) implies (5.6), since $c \geq 0$.

Thus, (5.1) $\wedge$ (5.4) implies (5.5). □

Furthermore, it can be observed, that for the α-SRDCST problem with $u_p \geq 0$ condition (5.1) implies condition (5.4). So for this special case of the problem, only (5.1) has to be checked.

5.1.1 Computational Issues

The paths which are used in the comparison can be found by solving a resource constrained shortest path problem. But this is known to be $\mathcal{NP}$-hard already for the deterministic case. There are pseudo-polynomial algorithms which could be used to find paths for the RDCST and the RRDCST problem, but for the SRDCST problem there may be no such algorithm.

The preprocessing should not be very time consuming. It is desirable that the sum of preprocessing and optimization time is smaller than the time needed for optimization, if the preprocessing phase is skipped. The comparison can be accelerated by only considering paths of length two. This could be done in polynomial time by iterating over all triangles in the graph.

5.2 Comparison with Root Arcs

An arc (u, v) can also be deleted, in case that taking the root arc (r, v) instead will always be at least equally good. Only if (u, v) and (v, u) are not needed to find an optimal solution, the edge $\{u, v\}$ can be deleted. This indicates that the preprocessing may be more successful if the reduction operates on the set A instead of E.

$$w_{rv} \leq w_{uv} \tag{5.8}$$

$$d_{rv} \leq d_u^{\min} + d_{uv} \tag{5.9}$$

In the deterministic case the conditions (5.8) and (5.9) describe the precondition. The variable $d_u^{\min}$ stands for the shortest (expected) delay of all paths from r to u and is defined as

$$d_u^{\min} = \min_{P(r,u)} d_{P(r,u)}. \tag{5.10}$$

RRDCST For the same reason as in Section 5.1 we have to check both (5.8) and (5.9) for the RRDCST problem. A bit more interesting is the scenario where the path P in a solution has a subpath P_{vt} from v to a terminal node t. The part between r and v consists of either $\{r, v\}$ or a path $P_{rv} = P(r, u) \cup \{u, v\}$. Let us assume the edge with the highest delay variation is $\{r, v\}$

for the first possible path and lies in P_{rv} for the other one. All other edges which influence $\hat{d}_P^\Gamma$ lie in P_{vt}. Then we would have to determine if

$$d_{rv} + \min(1, \Gamma) \cdot \hat{d}_{rv} \leq d_{P_{rv}} + \min(1, \Gamma) \cdot \max_{e \in P_{rv}} \{\hat{d}_e\}$$

holds. Since it has to be verified that there is no path for which the arc (u, v) could be of use, this has to hold for all possible paths P_{rv}. But it is enough to check not all paths, but only the path $P_{rv}^{\min}$ with the smallest delay $d_{P_{rv}}^{\Gamma'}$ calculated with $\Gamma' = \min(1, \Gamma)$.

$$d_{rv} + \min(1, \Gamma) \cdot \hat{d}_{rv} \leq d_{P_{rv}^{\min}} + \min(1, \Gamma) \cdot \max_{e \in P_{rv}^{\min}} \{\hat{d}_e\} \tag{5.11}$$

The problem of finding such a path is equivalent to the robust shortest path problem with uncertain weights. For this problem it was shown by Bertsimas and Sim [5] that it is still solvable in polynomial time. It is sufficient to verify (5.11) as third condition.

Theorem 5.3 *An arc (u, v) cannot be part of every optimal solution of a given instance of the RRDCST problem if* (5.8)*,* (5.9) *and* (5.11) *holds.*

Proof. Let us assume to the contrary that a directed path $P \subset A$, $(u, v) \in P$, from r to a terminal node t exists that is part of an optimal solution and using (r, v) instead of (u, v) increases the total weight or violates a delay constraint. But the total weight will not be higher because of (5.8). Let $P_{rv} \subset P$ be the subpath of P from r to v, $P_{vt} \subset P$ the subpath from v to t and $P_r = (r, v) \cup P_{vt}$. It follows from the assumptions that P_r would violate a delay constraint which means that $d_{P_r}^\Gamma > d_P^\Gamma$.

We distinguish two cases:

1. $\hat{d}_{rv}$ has no influence on $\hat{d}_{P_r}^\Gamma$. Adding all common delay variations to both sides of (5.9) leads to a contradiction.

$$d_{P_r}^\Gamma = d_{P_r} + \hat{d}_{P_{vt}}^\Gamma \leq d_P + \hat{d}_{P_{vt}}^\Gamma \leq d_P^\Gamma$$

2. $\hat{d}_{rv}$ influences $\hat{d}_{P_r}^\Gamma$ as a whole. Now we can add all common parts to both sides of inequality (5.11). Since $d_{P_{rv}^{\min}}$ has the minimal delay for $\Gamma = 1$, taking P_{rv} instead cannot decrease the delay. This leads to a lower bound for d_P^Γ which contradicts the assumption.

$$d_{P_r}^\Gamma = d_{P_{vt}}^{\Gamma-1} + d_{rv} + \hat{d}_{rv} \leq d_{P_{vt}}^{\Gamma-1} + d_{P_{rv}^{\min}} + \max_{e \in P_{rv}^{\min}} \{\hat{d}_e\} \leq d_{P_{vt}}^{\Gamma-1} + d_{P_{rv}} + \max_{e \in P_{rv}} \{\hat{d}_e\} \leq d_P^\Gamma$$

3. $\hat{d}_{rv}$ influences $\hat{d}_{P_r}^\Gamma$ fractionally. Therefore, we build an upper bound for d_{rv} as it was done in case 1.

$$d_{rv} \leq d_{P_{rv}}$$

Next, we build an upper bound for $d_{rv} + \hat{d}_{rv}$ analogous to case 2.

$$d_{rv} + \hat{d}_{rv} \leq d_{P_{rv}} + \max_{e \in P_{rv}} \{\hat{d}_e\}$$

Then we use a linear combination of these two inequalities which leads again to a contradiction.

$$d^{\Gamma}_{P_r} = d^{\lfloor\Gamma\rfloor}_{P_{vt}} + d_{rv} + (\Gamma - \lfloor\Gamma\rfloor) \cdot \hat{d}_{rv} \leq d^{\lfloor\Gamma\rfloor}_{P_{vt}} + d_{P_{rv}} + (\Gamma - \lfloor\Gamma\rfloor) \cdot \max_{e \in P_{rv}} \{\hat{d}_e\} \leq d^{\Gamma}_{P}$$

So the path P_r is part of an optimal solution without arc (u, v). □

SRDCST It holds also for the SRDCST problem, that constraints (5.8) and (5.9) have to be checked. Furthermore, the inequality

$$d_{rv} + u_p \cdot \check{d}_{rv} \leq d_{P_{rv}} + u_p \cdot \sqrt{\sum_{e \in P_{rv}} \check{d}^2_e}$$

has to be true $\forall P_{rv} = P(r, u) \cup \{u, v\}$. It would be enough if it holds for the path $P^{\min}_{rv}$ with the minimal delay $d^{p}_{P_{rv}}$.

$$d_{rv} + u_p \cdot \check{d}_{rv} \leq d_{P^{\min}_{rv}} + u_p \cdot \sqrt{\sum_{e \in P^{\min}_{rv}} \check{d}^2_e} \tag{5.12}$$

Theorem 5.4 *An arc (u, v) cannot be part of every optimal solution of a given instance of the SRDCST problem if* (5.8)*,* (5.9) *and* (5.12) *holds.*

Proof. Let us assume to the contrary that a directed path $P \subset A$, $(u, v) \in P$, from r to a terminal node t exists that is part of an optimal solution and using (r, v) instead of (u, v) increases the total weight or violates a delay constraint. But the total weight will not be higher because of (5.8). Let $P_{rv} \subset P$ be the subpath of P from r to v, $P_{vt} \subset P$ the subpath from v to t and $P_r = (r, v) \cup P_{vt}$.

We derive from (5.9) that $d_{rv} \leq d_{P_{rv}}$. Let c be the difference $d_{P_{rv}} - d_{rv}$. Then we derive from (5.12) that

$$d_{rv} + u_p \cdot \check{d}_{rv} \leq d_{P_{rv}} + u_p \cdot \sqrt{\sum_{e \in P_{rv}} \check{d}^2_e}.$$

Subtracting d_{rv} results in inequality (5.13)

$$u_p \cdot \check{d}_{rv} \leq c + u_p \cdot \sqrt{\sum_{e \in P_{rv}} \check{d}^2_e} \tag{5.13}$$

We denote the difference between the standard deviations as $\Delta\check{d} = |\check{d}_{rv} - \sqrt{\sum_{e \in P_{rv}} \check{d}^2_e}|$. It follows from the assumptions that P_r would violate a delay constraint which means that $d^{p}_{P_r} > d^{p}_{P}$. This can also be stated as

$$d_{P_{vt}} + u_p \cdot \sqrt{\check{d}^2_{rv} + \sum_{e \in P_{vt}} \check{d}^2_e} > d_{P_{vt}} + c + u_p \cdot \sqrt{\sum_{e \in P_{rv}} \check{d}^2_e + \sum_{e \in P_{vt}} \check{d}^2_e}.$$

We subtract $d_{P_{vt}}$ and use $c' = \sum_{e \in P_{vt}} \check{d}_e^2$.

$$u_p \cdot \sqrt{\check{d}_{rv}^2 + c'} > c + u_p \cdot \sqrt{\sum_{e \in P_{rv}} \check{d}_e^2 + c'} \tag{5.14}$$

Since $c' \geq 0$, the new difference of the standard deviations can only decrease.

$$\Delta\check{d} \geq |\sqrt{\check{d}_{rv}^2 + c'} - \sqrt{\sum_{e \in P_{rv}} \check{d}_e^2 + c'}|$$

But this means that inequality (5.13) contradicts with (5.14), since $c \geq 0$.

So the path P_r is part of an optimal solution without arc (u, v). □

It is very likely that the problem of finding the minimal path $P_{rv}^{\min}$ is not solvable in polynomial time anymore. Only for $u_p \geq 0$ there is the possibility for a faster comparison by a more restrictive condition (5.15). With this approximation it is not possible to find all dispensable arcs for every instance.

$$d_{rv} + u_p \cdot \check{d}_{rv} \leq d_u^{\min} + d_{uv} + u_p \cdot \check{d}_{uv} \tag{5.15}$$

The α-SRDCST problem is an interesting special case for this comparison. For the case $u_p \geq 0$ we get the result that (5.15) implies (5.9). The case $u_p \leq 0$ leads even to the fact that (5.9) implies (5.12) which allows a very efficient comparison.

5.3 Infeasible Arcs and Nodes

Sometimes it is impossible that an arc $(u, v) \in A$ appears in a feasible solution, because there is no path P that contains (u, v) and fulfils the delay constraints. The computation that is necessary to decide if this holds is analogous to Section 5.2. Because of the high complexity it might be beneficial to apply this test in a heuristic form. For the RRDCST problem this can be done by checking inequality (5.16) and for the SRDCST problem with $u_p \geq 0$ by checking inequality (5.17). For the SRDCST problem with $u_p < 0$ even effective heuristics are hard to find. The delay $d_u^{\min}$ is again defined with the term (5.10). The terms on the left sides of the inequalities are lower bounds of the delay d_P^{Γ} and d_P^{p}, respectively, of every path P from r to v that contains (u, v). Thus, (u, v) cannot be part of any feasible solution.

$$d_u^{\min} + d_{uv} + \min(1, \Gamma) \cdot \hat{d}_{uv} > B \tag{5.16}$$

$$d_u^{\min} + d_{uv} + u_p \cdot \check{d}_{uv} > B \tag{5.17}$$

Similarly, a potential Steiner node cannot be part of an optimal solution if it is only possible as a leaf or cannot be used in a feasible solution at all. Infeasible arcs and nodes do not appear regularly, but only if B is very small in comparison to the edge delays or if the graph is very sparse.

CHAPTER 6

Instance Transformations

For the robust and the stochastic problem there is a well-founded concept for dealing with uncertain delays. But there is still the question if we really need to extend the deterministic problem. Instead, the problem instances could be altered by either changing the edge delays or the delay bound to achieve similar effects. Observe, that the factor α of α-RRDCST and α-SRDCST problem reduces the amount of information which has to be included in the instance. Without this factor there is also information of the delay variations or standard deviations needed. Since instances of the deterministic problem contain less data of edge delays, most instances of the RRDCST and the SRDCST problem cannot be transformed to a deterministic instance without loss. However, a small loss may still lead to an adequate solution. For the worst case in the RRDCST problem there is a lossless transformation, where all delays are set to the worst case already at the instance, i.e., $d'_e = d_e + \hat{d}_e, \forall e \in E$. We will further discuss transformations for bad cases that are in general not lossless.

6.1 Altering the Delay Bound

Inequality (6.1) is a simplified version of the delay constraints of the RRDCST and the SRDCST problem.

$$\sum_{e \in P_{G'}(r,t)} d_e + c_{P_{G'}(r,t)} \leq B \qquad \forall t \in T \tag{6.1}$$

We can observe for bad cases that some term $c_{P_{G'}(r,t)} > 0$ is added to the expected value. For the decision whether a path is feasible or not, paths with a delay d_P^Γ for the RRDCST problem and d_P^p for the SRDCST problem close to B are most likely to get misclassified. Such paths will often consist of edges with similar delay variations or standard deviations, respectively, because the probability that one path contains only high values and another one only low values is very low, at least for randomly generated instances. Therefore, the terms $c_{P_{G'}(r,t)}$ may lie in a similar range. So considering this term as constant for the whole instance may be a good approximation. But

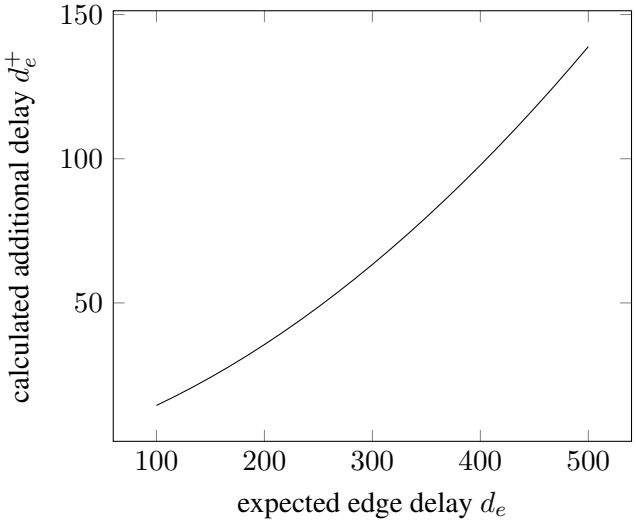


Figure 6.1: A possible non-linear function that would prefer short delays and thus long paths.

then we could subtract $c_{P_{G'}(r,t)}$ from B and solve the corresponding deterministic problem. This transformation leads to more solutions that would be suboptimal or infeasible for the RRDCST or the SRDCST problem, the more paths with a delay d_P^Γ or d_P^p close to B vary in the number of edges.

6.2 Altering Edge Delays

The delay bound is just one number. Changing the edge delays would give us much more flexibility for an instance transformation. Let us first observe another effect of the two problems.

Let us assume there is a path P_1 with few edges and a path P_2 with a lot of edges. Furthermore, all edges in the same path have equal delays d_1 and d_2, respectively. For the α-RRDCST problem we would get delay constraints of the form (6.2).

$$|P_i| \cdot d_i + \alpha \cdot \min(|P_i|, \Gamma) \cdot d_i \leq B \qquad i \in \{1, 2\} \tag{6.2}$$

We consider the case where both paths have the same expected delay, i.e., $|P_1| \cdot d_1 = |P_2| \cdot d_2$. It follows directly that

$$\min(|P_1|, \Gamma) \cdot d_1 \geq \min(|P_2|, \Gamma) \cdot d_2.$$

The case $\Gamma = 0$ which is equivalent to the deterministic problem would either classify both paths as feasible or both as infeasible. But with $\Gamma > 0$, P_2 is more likely to be feasible than P_1. In general it could be said that paths with more edges are preferred over shorter ones. This effect occurs also in the α-SRDCST problem. Since long paths are only possible with short edge delays, we could transform the instance such that high delays are handled more pessimistic. This could be done with a function like in Fig. 6.1.

With this approach we could use an existing implementation of the deterministic problem if it allows delay values to be in $\mathbb{Q}^+$. Then it would only be necessary to transform the instance with a suitable function. However, such a transformation cannot be used to solve the α-RRDCST or the α-SRDCST problem exactly. The problem that is actually solved differs from both and its description is not very intuitive. Nevertheless, the quality of the results can be good and depends highly on the used function. The following part will demonstrate how such a function could be found.

One possibility is to approximate the normally distributed approach. This results in the task of converting the term $\sqrt{\sum_e \check{d}_e^2}$ into a term of the form $\sum_e k_e \cdot \check{d}_e$. This cannot be done without a small error. But let us assume all edge delays in the same path are equal. Let m be the maximal number of edges such that the path is feasible. Then the following formula would hold:

$$B - m \cdot d_e = u_p \cdot \sqrt{m \cdot \check{d}_e^2}$$

After solving the quadratic equation we get two solutions.

$$m_1 = \frac{u_p^2 \cdot \check{d}_e^2 + 2 \cdot B \cdot d_e - \sqrt{u_p^4 \cdot \check{d}_e^4 + 4 \cdot B \cdot d_e \cdot u_p^2 \cdot \check{d}_e^2}}{2d_e^2}$$

$$m_2 = \frac{u_p^2 \cdot \check{d}_e^2 + 2 \cdot B \cdot d_e + \sqrt{u_p^4 \cdot \check{d}_e^4 + 4 \cdot B \cdot d_e \cdot u_p^2 \cdot \check{d}_e^2}}{2d_e^2}$$

It holds that $m = m_1$ if $u_p \geq 0$ and $m = m_2$ if $u_p < 0$. The last step is to split the standard deviation of the whole path into equal parts. As a final result we get the additional delay

$$d_e^+ = u_p \cdot \frac{\sqrt{m \cdot \check{d}_e^2}}{m}, \forall e \in E$$

for the function we were looking for. Unfortunately, the result tends to be too low for paths with different edge delays. This can be seen in the following example.

We have an instance of the 0.5-SRDCST problem with $u_p = 1$ and $B = 600$ and we want to determine if path P is feasible. Table 6.1 contains information of all edges of P. There are 160 edges with an expected delay of 2 and one edge with an expected delay of 200. With this information we can verify if the delay constraints hold.

Table 6.1: Edge information of path P.

#	d	$\hat{d}$	$\sum d$	m	d^+	$\sum d^+$
160	2	1	320	291.5	0.059	9.4
1	200	100	200	2.25	66.7	66.7
			520			76

$$d_P^p = d_P + u_p \cdot \sqrt{160 \cdot 1^2 + 100^2} = 520 + 100.8 = 620.8$$

$$d_P + \sum_{e \in P} d_e^+ = 520 + 76 = 596$$

Since $d_P^p > B$, P is infeasible for the 0.5-SRDCST problem. But for the RDCST problem with transformed edge delays P is feasible. This is an example where a path is misclassified. Since this function leads usually to a very good approximation of d_P^p, optimal solutions will be equivalent to the α-SRDCST problem for many instances.

6.3 Limitations of Instance Transformation

As we have seen in this chapter, existing algorithms for the RDCST problem can be reused to solve new problem variants by transforming the instances. Since instances of the RDCST problem can usually be solved faster, the question arises what is gained with algorithms for the RRDCST or the SRDCST problem. Instance transformations are easier to implement, but they also have some drawbacks.

- Instance transformations can be used to solve the RRDCST or the SRDCST problem approximately, where some solution are wrongly classified as feasible or infeasible. However, this approach is not generally applicable, because for some problems it isn't predetermined which result of a random variable is the worst value. One can think of a variation of the RDCST problem, where there are additional to the upper delay bound constraints for a lower delay bound. Then, for a single edge it cannot be said anymore whether a short or a long delay is beneficial.

- Especially the transformation of the edge delays is a good approximation of the delay constraints of the α-SRDCST problem. But the quality of the approximation of both presented instance transformations can be much worse if we want to solve instances of the RRDCST or the SRDCST problem without the constant α.

- Even if the solutions seem to be good for a set of instances, it will be hard to make precise statements about their quality. There is a well-founded and simple concept of optimality for the SRDCST problem. Also for the RRDCST problem the definition of optimality is easy understandable. With instance transformations the concept of optimality is a bit more artificial and it may be hard to derive important probabilistic results from an optimal solution.

CHAPTER 7

Results

7.1 Instances

For almost all tests the problem instances from Gouveia et al. [25] are used. They define complete undirected graphs either with 21 or 41 nodes including root r with edge weights and delays. The bound B is separately added as a parameter in the implementation to provide a higher flexibility. Originally, the instances were not intended for Steiner tree problems. Therefore another parameter is introduced to define the number of terminal nodes. The first $|T|$ non-root nodes in the instance define set T.

The uncertainties are included with factor α, and depending on the problem either with Γ or u_p. This way, only the special problems α-RRDCST and α-SRDCST are solved which seem to be the more realistic cases in real world.

The instances have the following naming scheme:

$$\langle \text{type} \rangle \langle |V \setminus \{r\}| \rangle - \langle i \rangle - \text{wa} \langle d_{\max} \rangle$$

There are three types of instances for which the edge weights were generated differently. The weights are always numbers between 1 and 100. The delays are random numbers within the range 1 to $d_{\max}$. For the type *tr* the weights are also generated randomly. For the other two types the weights follow from node positions in a two-dimensional space. So they are Euclidean. The *tc*-instances have their root node centred in the space whereas the root node is placed in a corner for *te*-instances. The number of the instance i goes from 1 to 5. So there are always five instances which were generated upon the same rules.

Instances with more nodes were generated by Ruthmair [44]. These instances contain complete graphs with uniformly distributed edge weights and delays, both in the interval from 1 to 99.

7.2 Implementations

All implementations are written in C++ and use IBM ILOG CPLEX 12.5 as optimizer. For the performance tests Intel Xeon processors E5540 with 2.53 GHz and eight cores in combination with 24 GB RAM and also processors of the type E5649 with 2.53 GHz, twelve cores and 60 GB RAM were used. All implementations use only one thread and thus only one core is utilized.

There are some different implementations where M can only solve the RRDCST problem, whereas the others can solve both the RRDCST and the SRDCST problem.

- M: a branch-and-bound algorithm based on the MCF formulation
- PC: a branch-and-cut algorithm based on the path-cut formulation where cuts are added only on integral solutions
- PC2: a branch-and-cut algorithm based on the path-cut formulation where cuts are added on fractional and integral solutions
- LG: a branch-and-cut algorithm based on the layered graph formulation where cuts are added on fractional and integral solutions
- LGCC: LG with connection cuts (3.40). Their separation is based on [44].

7.2.1 Measurement

When comparing the different implementations, the five numbered instances are always viewed together. Each instance has a time limit of 10,000 seconds and a memory limit of 3 GB. The different properties of the processes are then evaluated as explained below.

Time The solving time is measured in seconds. If the program is aborted because of the time or the memory limit, the solving time will be assumed to be an unknown number greater than 10,000. Here, always the median is given which corresponds to the third lowest running time. If less than three instances are solved to proven optimality, the median cannot be computed. All values are rounded to whole seconds. A median of 0.4 is therefore given as 0.

Solved Instances An instance counts as solved if an optimal solution is found and its optimality is proven within time and memory limits. The number of solved instances is given which can take values from 0 up to the total number of instances, which is 5 for the instances from Gouveia and 30 for the instances from Ruthmair. Additionally, for the RRDCST problem an optimal solution could also be feasible in the worst case. The number of the instances for which an optimal worst case solution was found is given in parentheses or omitted if equal to zero. It is always less than or equal to the number of solved instances.

Gap If it cannot be proven that the best solution which was found is optimal, it may be interesting how close this solution is to the optimum. Unfortunately, the optimum is in general not known. But if a lower bound for the optimum is known, a gap can be calculated as

$$1 - \frac{\text{lower bound}}{\text{value of the best known solution}}.$$

If no feasible solution is found the gap is defined as 1 which is the worst possible value. A gap of 0 means that the optimality was proven. The gap is given in percent as an average over the five instances.

Branch-and-Bound Nodes The number of nodes is counted after the termination of the process. There are three possible reasons for termination which are a solution with proven optimality and exceeded time or memory limits. The number of nodes is given as an average over the five instances. A higher average is usually worse, but the other three criteria are assumed to be more important. The lowest value is therefore not highlighted in the tables. In some cases there is no need to construct a branch-and-bound tree at all and thus the number of nodes is 0.

7.3 Comparison of Solutions

There are difficulties in comparing the different methods of dealing with uncertain delays. In the robust approach there is the assumption of a known worst case, whereas in the stochastic approach in theory delays can get arbitrarily high. Moreover, the robust approach makes no assumptions about the probability distributions, except that it is symmetric and values lie within a bounded interval, but for each of them the results have a different quality. Nearly the same holds for the stochastic method. The difference is that the knowledge of the standard deviations and approximately normally distributed delays lead to a very natural concept of optimality in the context of uncertainty. A detailed comparison of the two approaches is best made for a given probability distribution. The goal of this comparison is not to find the approach which is better than the other, but to show differences and similarities of optimal solutions to instances of the different problems.

Here, we consider the following scenario. For the α-RRDCST problem we use the factor $\alpha = 0.5$. The edge delays are normally distributed which violates the assumption of the robust approach that there is a finite worst case. Clearly, a common scenario has to violate at least one assumption, because the assumption of a normal distribution contradicts with the assumption of a bounded interval. The factor α of the α-SRDCST is set to $\frac{0.5}{u_{0.95}} = 0.304$. The constant $u_{0.95} = 1.6449$ is the 0.95-quantile of the normal distribution and should not be confused with the p-quantile which is part of an instance of the SRDCST problem. As it can be seen in Figure 7.1, this gives us a probability of 5% that the delay which is assumed to be the worst case could be exceeded. This scenario gives us values of p which are also meaningful for solutions of the robust problem. Because good cases are not treated in robust problems, they are omitted in the comparison of the solutions of the two problems.

One might think that the choice of $u_{0.95}$ affects the whole computation, but it can be shown that it has no effect except a scaling of u_p. To check whether a delay constraint is fulfilled, the

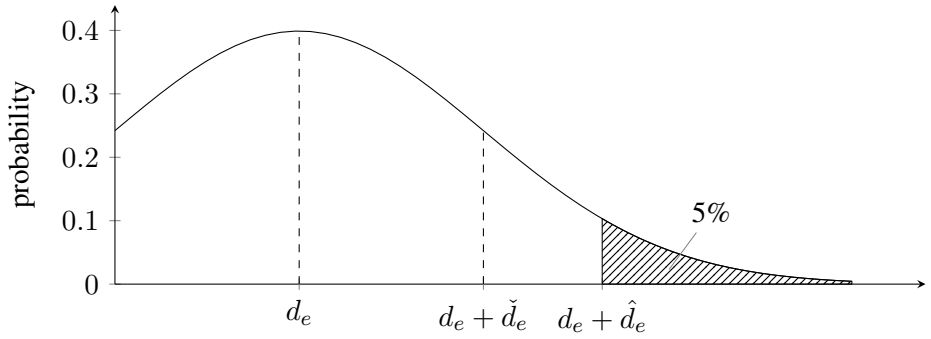


Figure 7.1: Probability distribution of the delay of an edge e. The delay is normally distributed with mean d_e and standard deviation $\check{d}_e$. Furthermore, $\hat{d}_e = u_{0.95} \cdot \check{d}_e$. As a consequence, there is a probability of 5% that the defined worst case $d_e + \hat{d}_e$ is exceeded.

result of term (7.1) has to be calculated.

$$d_P^p = d_P + u_p \cdot \sqrt{\sum_{e \in P} (\alpha \cdot d_e)^2} \tag{7.1}$$

$$= d_P + u_p \cdot \sqrt{(2 \cdot \alpha)^2 \cdot \sum_{e \in P} (0.5 \cdot d_e)^2} = d_P + 2 \cdot \alpha \cdot u_p \cdot \sqrt{\sum_{e \in P} (0.5 \cdot d_e)^2} \tag{7.2}$$

An arbitrary factor can be moved outside the square root. This means that every α-SRDCST problem can be transformed into an equivalent 0.5-SRDCST problem by using $u'_p = 2 \cdot \alpha \cdot u_p$ as quantile of the transformed instance.

Here, the parameters Γ or u_p are not included in a problem instance. The goal is to solve both problems in a bi-objective manner instead. Basic concepts of bi-objective optimization are described in Section 2.4. The Pareto optimal solutions are determined with an epsilon-constraint method. The first problem which is solved is the expected case with $\Gamma = 0$ and $u_p = 0$, respectively. For all paths in the solution there is an associated maximal value of the parameter. The minimum of those values is the maximal value of Γ or u_p for which the solution is still feasible. We can then find the next solution by adding a small number $\epsilon > 0$ to the parameter and solving the next optimization problem.

This technique gives us a set including all Pareto optimal solutions. Figure 7.2 shows such a Pareto front for one instance. Note that the illustrated function has to be monotonically increasing. Sometimes solutions with equal weight are found with the epsilon-constraint method, but clearly only one of them can be Pareto optimal. It can be seen that a higher probability p has its price, which is usually referred to as the price of robustness. This term was coined by Sim [48].

It is possible to extract different properties from all solutions. Obviously we have the corresponding weight. The knowledge of the worst edge delays gives us also the maximal Γ under which a solution is still feasible. This can also be done for solutions of the SRDCST problem. Analogously, the maximal value of u_p and the corresponding probability p can be calculated.

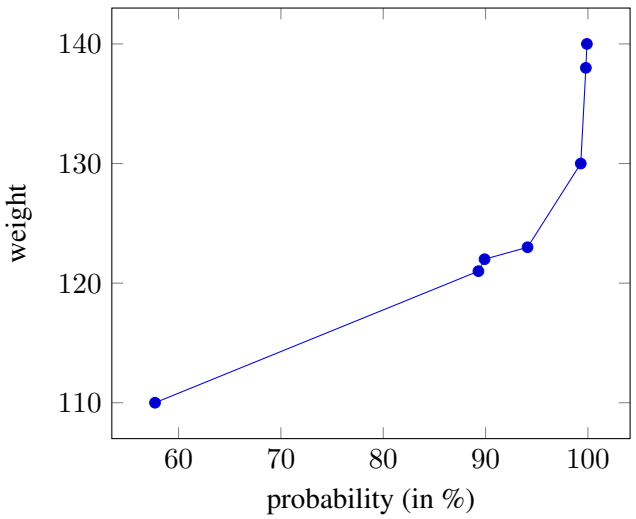


Figure 7.2: Pareto optimal solutions of the α-SRDCST problem for instance tr20-1-wa100 with |T|=10 and B=200 with respect to u_p and weight of the solution. Every value u_p corresponds directly to a value of the probability p.

A further interesting property is the expected delay of the solution which corresponds to the smallest possible value of B in the deterministic problem.

In Table 7.1 these properties are shown for different solutions of three exemplary problem instances. Every solution was found by at least one of the following three implementations. The used algorithms are based on the path-cut formulation from Section 3.2.2 and Section 4.3.2.

- S: an algorithm that solves the bi-objective variant of the α-SRDCST problem where u_p is not given explicitly in an instance and should be maximized.
- R: an algorithm that solves the bi-objective variant of the α-RRDCST problem where Γ is not given explicitly in an instance and should be maximized.
- D: an algorithm that solves the bi-objective variant of the RDCST problem where an upper bound for B is given and B should be minimized. In the context of uncertain delays with a symmetric probability distribution D finds all Pareto optimal solutions with respect to weight and expected delay.

For R all Pareto optimal solutions are found and there are also some which are not Pareto optimal but part of the Pareto front. For S and D there are also Pareto optimal solutions with very low or very high total weight which are not given in the table. Note, that basically the same instance can be given to S, R and D without any transformation. Surprisingly, the solutions of the instances of the three problems are very similar. In this setting solutions of the SRDCST problem are the most promising ones, because the edge delays are normally distributed.

The first Pareto optimal solution that is found with each algorithm is an optimal solution of the expected case. We know that for such a solution each constraint has to be valid with at least

Table 7.1: A list of found solutions for three different instances. S is an implementation of the α-SRDCST problem, R an implementation of the α-RRDCST problem and D an implementation of the RDCST problem where B is iteratively decreased.

instance	weight	max. Γ	max. u_p	p (in %)	expected delay	S	R	D
tr20-1-wa100 |T|=10, B=200	110	0.165	0.194	57.7	192	x	x	x
	121	0.887	1.240	89.3	157	x	x	x
	122	0.884	1.274	89.9	158	x		
	123	1.137	1.561	94.1	148	x	x	x
	130	2.565	2.462	99.3	143	x	x	x
	138	3.917	2.873	99.8	136	x	x	x
	139	∞	2.873	99.8	114		x	x
	140	∞	3.180	99.9	123	x		
tc20-2-wa1000 |T|=15, B=1500	345	0.192	0.223	58.8	1449	x	x	x
	351	0.258	0.386	65	1402	x	x	x
	354	0.367	0.542	70.6	1363	x	x	x
	356	0.344	0.532	70.3	1345			x
	358	0.357	0.559	71.2	1339	x		x
	359	0.690	1.108	86.6	1196	x	x	x
	361	0.690	1.108	86.6	1189			x
	365	1.254	1.519	93.6	1196	x	x	
	366	0.906	1.435	92.4	1156			x
	370	1.781	1.751	96	1178	x	x	
	372	2.842	1.840	96.7	1029	x	x	
	372	2.842	2.488	99.4	1029	x		x
	379	∞	2.535	99.4	914	x	x	x
tr40-4-wa100 |T|=20, B=240	52	0.625	0.875	80.9	210	x	x	x
	53	0.646	0.903	81.7	209	x	x	x
	55	0.864	1.130	87.1	205	x	x	x
	56	1	1.414	92.1	192	x	x	x
	57	1.261	1.581	94.3	192	x	x	
	59	1.261	1.581	94.3	188			x
	67	1.25	1.593	94.4	192	x		
	67	1.75	1.787	96.3	192	x	x	
	68	1.769	1.787	96.3	192		x	
	68	1.846	1.794	96.4	190	x	x	
	69	1.25	1.593	94.4	187			x
	69	1.75	1.895	97.1	186	x		
	69	1.75	1.895	97.1	182			x
	70	2.306	2.138	98.4	186	x	x	
	70	3.571	2.524	99.4	164	x	x	x
	71	∞	2.719	99.7	159	x	x	x

50%. It can be seen in Table 7.1 that for a given solution this probability is often higher. For the third expected case solution there is even a chance of 80.9% for each constraint to be valid. It should also be observed that a higher value of Γ can correspond to a lower value of u_p and thus a lower probability p, which is in fact the reason that the Pareto fronts of S and R differ. For the three instances there are only small differences between the three Pareto fronts. The table shows also a small contradiction which is caused by the assumption that there is no finite worst case. Then, a worst case solution of the α-RRDCST problem has a corresponding p which is slightly smaller than 100%. The impact of this contradiction on the quality of the results of R is only small.

The transformation presented in Section 6.2 would result in nearly the same set of Pareto optimal solutions as implementation S. The drawback is that the complex mathematical terms used in this transformation complicate bi-objective implementations.

7.4 Performance Test

For $\mathcal{N}P$-hard problems there is usually the question, how fast they can be solved in practice and for which size the instance is still solvable within a reasonable amount of time. Most of the smaller instances with 21 nodes can by solved within a few seconds. The instances with 41 nodes are already hard to solve. This holds for both the α-RRDCST and the α-SRDCST problem. But there are huge differences for the different formulations described in Section 3.2 and 4.3.

With CPLEX there are two possibilities of adding new constraints during the optimization process. *Lazy constraints* can only be added if an integral solution is found for the smaller model. This way it is possible to add only a few constraints at the beginning and others only in case of violations. With *user cuts* violated constraints can also be added on fractional solutions of the corresponding linear program. This gives us two possibilities for a branch-and-cut algorithm:

- Use only lazy constraints.
- Use lazy constraints and user cuts.

Table 7.2 compares these two strategies for the path-cut formulation. The additional user cuts don't seem to improve the performance as it can be seen in Fig. 7.3a. One possible explanation for this is that because of the weakness of the formulation a massive branching is necessary for many instances but delayed by the user cuts. Table 7.3 shows the results for the te-instances where no implementation outperforms the other. For the instances with $|T| = 5$ the user cuts increase the performance. An important difference between the two strategies is that additional user cuts often reduce the number of branch-and-bound nodes and also the needed memory. If only 1 GB was available, more instances could not be solved if only lazy constraints are used.

There are also different strategies for the layered graph approach. The performance of the implementations LG and LGCC is compared for instances with $B \leq 240$ in Tables 7.4–7.7. Overall LGCC performs slightly better which can be seen in Fig. 7.3b. The disadvantage is that if no optimal solution is found, the best feasible solution will have a large gap to optimality and in many cases no feasible solution will be found at all. The additional connection cuts lead to

a very strong formulation which can be observed in the very low number of branch-and-bound nodes.

Table 7.8 compares the three different formulations on the same set of instances. Only the path-cut implementation with lazy constraints (PC) is used for the comparison from now on. The layered graph implementation (LG) on the other hand makes use of the user cuts. The impact of the user cuts on this implementation is only small, because there are in general not as many constraint violations. Let us first analyse the performance of the implementation that uses an MCF formulation (M). There is a strong correlation between the number of terminal nodes $|T|$ and the running time, see also Fig. 7.5b for the whole set of instances. For a smaller value of $|T|$ there are less flows to be constructed which causes the set of variables to be smaller. Furthermore, a small correlation with Γ can be seen in the table and similarly in Fig. 7.5a. The case $\Gamma = 0$ renders the constraints (3.22) trivial, because every variable q_k is then allowed to be arbitrarily high. As a consequence, all variables p_{ij}^k can be set to 0. The implementation LG shares this correlation, see Fig. 7.9a, but the reason is a different one. If $\Gamma = 0$, there are no path-cuts that can be violated. A higher value of Γ means that more infeasible paths are in the layered graph. But this effect is compensated by the reduction of layers. It can be seen on the instances with $\Gamma = 4.5$ that this reduction is very important, because they are often solved faster than the instances with $\Gamma = 3$. There is always a point where increasing Γ has no effect on the optimal solution, because it is already feasible in the worst case. In Table 7.8 this can be seen for the instances with $|T| = 5$ and $\Gamma = 3$. All found solutions for these instances are already feasible in the worst case.

In Tables 7.9 and 7.10 the value of $d_{\max}$ is changed from 10 to 100 and 1000, respectively. There are different results for all three implementations. For M and PC there is no direct correlation of the running time and $d_{\max}$ which can be seen in Fig. 7.6a and 7.8a. The differences are just caused by the specific delays, because they were created separately. For LG such a correlation exists which is shown in Fig. 7.10a. The number of layers increases with higher values of B. A further consequence is a larger layered graph with more nodes and more arcs. While LG is often the best option for solving instances with $B = 12$ it performs definitely worse on $B \geq 120$.

Tables 7.11, 7.12 and 7.13 summarize the performance for te-instances. These are harder to solve than tr-instances which is already indicated in Chapter 5. Only for $B = 12$, LG is the most successful implementation, while PC dominates for larger values. However, most of the difficult instances with $|T| = 35$ and $B = 2.4 \cdot d_{\max}$ cannot be solved to proven optimality within 10000 seconds. The characteristic differences of the running times on the whole instance set are visualized in the Figures 7.4–7.10 with cumulative frequency diagrams. The correlation between $|T|$ and the running time can not only be observed for M, but also for PC and LG in Fig 7.7b and Fig 7.9b. So instances with less terminal nodes are usually easier to solve. More interesting is the correlation between the value of $\frac{B}{d_{\max}}$ and the running time. There we have the group of instances where B is relatively low to $d_{\max}$ which is easier to solve for PC, see Fig. 7.8b. Figure 7.10b shows a more extreme result for LG, but this is mainly caused by the smaller values of B. In Fig. 7.6b we see that this group of instances is harder to solve for M.

Table 7.14 shows results for the 0.5-SRDCST problem. As already mentioned, the implementation M is not suited to solve the problem, because constraints (4.8) of the problem formu-

lation are not linear. For PC and LG both problems are solved analogously and the performance is also very similar. This similarity is no coincidence, since the main algorithms are the same for both problems. One difference is the feasibility check of a path, which is called as a subroutine in the separation method. It holds for the RRDCST and the SRDCST problem that the feasibility check has a relatively low complexity in comparison to the whole algorithm. The similarity of the results be seen in a comparison with Table 7.11. For LG there is also the difference in the reduction of layers which may be the reason that more instances with $|T| \geq 20$ and $B = 24$ are solved for the 0.5-SRDCST as for the 0.5-RRDCST problem. But this is also a minor difference, since both reductions follow the same concept. Because of the similarity of the results for bad cases, the performance test is omitted for the rest of the instance set. Good cases are not the focus of this thesis, but they are listed in Table 7.14 for the sake of completeness. With $u_p = -1$ there is a probability of at least 16% for every constraint to be valid. For PC there are only minor differences when working with $u_p < 0$. For LG there have to be additional layers and the number of infeasible paths in the layered graph increases. It can be observed that the instances with $u_p = -1$ are much harder to solve than the instances with $u_p \geq 0$.

All instances considered so far contain complete graphs which are still dense after deleting some edges in the preprocessing. The instance files from Ruthmair with 100 nodes were used in order to examine also graphs which are rather sparse. The delay bound B is set clearly below the maximal edge delay $d_{\max} = 100$. This way, a lot of edges can be deleted in the preprocessing. With $B = 20$ and $\Gamma > 0$ there are already instances without any feasible solution, therefore the performance test is done for $B = 25$ and $B = 40$. Table 7.15 summarizes the results on these instances for the 0.5-RRDCST problem. Basically, the differences in the running times are similar to the tests with dense graphs. Differently to the dense graphs, M performs better for the smaller value of B. This is because B has the side effect of controlling the density of the graph. Therefore the instances with $B = 25$ are definitely easier to solve than those with $B = 40$. A bit surprising are the relatively bad results of LGCC in comparison to LG for instances with $\Gamma > 0$ if we compare them with Fig. 7.3b. Unfortunately, if a combination of path cuts and connection cuts is used, it gets very difficult to find explanations for such unexpected effects.

Table 7.2: Results for instance files tr40-i-wa10, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. The implementation PC adds only lazy constraints with the separation methods. The alternative implementation PC2 adds additionally violated constraints as user cuts. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)		solved (worst case)		average gap (in %)		average number of B&B-nodes	
$\|T\|$	B	Γ	PC	PC2	PC	PC2	PC	PC2	PC	PC2
5	12	0	**1**	**1**	**5**	**5**	**0**	**0**	850	883
		1.5	**1**	**1**	**5** (2)	**5** (2)	**0**	**0**	442	350
		3	**1**	**1**	**5** (5)	**5** (5)	**0**	**0**	1222	579
		4.5	**1**	**1**	**5** (5)	**5** (5)	**0**	**0**	1262	602
5	24	0	**1**	**1**	**5** (1)	**5** (1)	**0**	**0**	17	23
		1.5	**1**	**1**	**5** (3)	**5** (3)	**0**	**0**	568	1792
		3	**1**	**1**	**5** (5)	**5** (5)	**0**	**0**	920	1148
		4.5	**1**	3	**5** (5)	**5** (5)	**0**	**0**	1033	1291
20	12	0	3	**2**	**5**	**5**	**0**	**0**	1231	922
		1.5	**1**	2	**5**	**5**	**0**	**0**	1304	626
		3	**1**	3	**5** (5)	**5** (5)	**0**	**0**	1577	745
		4.5	**1**	4	**5** (5)	**5** (5)	**0**	**0**	1499	692
20	24	0	**2**	7	**5**	**5**	**0**	**0**	1436	718
		1.5	**3**	18	**5**	**5**	**0**	**0**	1823	2312
		3	**5**	29	**5** (4)	**5** (4)	**0**	**0**	5708	3325
		4.5	**3**	11	**5** (5)	4 (4)	**0**	3.4	3986	5286
35	12	0	**2**	6	**5**	**5**	**0**	**0**	1195	2076
		1.5	**1**	10	**5**	**5**	**0**	**0**	364	438
		3	**2**	**2**	**5** (2)	**5** (2)	**0**	**0**	435	248
		4.5	**1**	5	**5** (5)	**5** (5)	**0**	**0**	800	218
35	24	0	**5**	20	**5**	**5**	**0**	**0**	2124	1215
		1.5	**4**	13	**5**	**5**	**0**	**0**	2117	1396
		3	**4**	61	**5**	**5**	**0**	**0**	2166	3938
		4.5	**8**	97	**5** (5)	**5** (5)	**0**	**0**	4528	3049

Table 7.3: Results for instance files te40-i-wa10, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. The implementation PC adds only lazy constraints with the separation methods. The alternative implementation PC2 adds additionally violated constraints as user cuts. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)		solved (worst case)		average gap (in %)		average number of B&B-nodes	
$\|T\|$	B	Γ	PC	PC2	PC	PC2	PC	PC2	PC	PC2
5	12	0	8	**2**	**5**	**5**	**0**	**0**	2184	203
		1.5	3	**1**	**5** (4)	**5** (4)	**0**	**0**	720	23
		3	2	**1**	**5** (5)	**5** (5)	**0**	**0**	1114	29
		4.5	2	**1**	**5** (5)	**5** (5)	**0**	**0**	562	29
5	24	0	3	**1**	**5**	**5**	**0**	**0**	1943	0
		1.5	3	**1**	**5**	**5**	**0**	**0**	1085	56
		3	6	**1**	**5** (4)	**5** (4)	**0**	**0**	1531	40
		4.5	8	**2**	**5** (5)	**5** (5)	**0**	**0**	1908	65
20	12	0	**93**	102	**5**	**5**	**0**	**0**	14398	3533
		1.5	**6**	14	**5**	**5**	**0**	**0**	2676	1645
		3	**7**	20	**5** (5)	**5** (5)	**0**	**0**	3997	2356
		4.5	**9**	16	**5** (5)	**5** (5)	**0**	**0**	3728	2968
20	24	0	**35**	59	**5**	**5**	**0**	**0**	18540	6150
		1.5	433	**327**	**5**	**5**	**0**	**0**	53165	23395
		3	**363**	634	4 (1)	**5** (1)	0.7	**0**	53050	23836
		4.5	**565**	1037	**5** (3)	**5** (3)	**0**	**0**	88920	25399
35	12	0	**83**	298	4	**5**	0.7	**0**	29023	11667
		1.5	**8**	20	**5**	**5**	**0**	**0**	1826	757
		3	**10**	17	**5** (3)	**5** (2)	**0**	**0**	1488	734
		4.5	**13**	24	**5** (5)	**5** (5)	**0**	**0**	2557	873
35	24	0	-	-	0	0	5.2	**5.2**	116764	105125
		1.5	-	-	**1**	**1**	**4.5**	6.4	88511	66234
		3	-	-	**1**	**1**	5.5	**3.9**	109205	96803
		4.5	-	-	0	0	**5.1**	5.3	118491	96789

Table 7.4: Results for instance files tr40-i-wa10, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. The implementations LG and LGCC are based on the layered graph formulation where LGCC uses connection cuts. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)		solved (worst case)		average gap (in %)		average number of B&B-nodes	
$\|T\|$	B	Γ	LG	LGCC	LG	LGCC	LG	LGCC	LG	LGCC
5	12	0	**1**	**1**	**5**	**5**	**0**	**0**	0	0
		1.5	**1**	**1**	**5** (2)	**5** (2)	**0**	**0**	0	0
		3	**1**	**1**	**5** (5)	**5** (5)	**0**	**0**	19	6
		4.5	**1**	**1**	**5** (5)	**5** (5)	**0**	**0**	0	1
5	24	0	**2**	**2**	**5** (2)	**5** (2)	**0**	**0**	0	0
		1.5	**4**	5	**5** (3)	**5** (3)	**0**	**0**	191	13
		3	5	**3**	**5** (5)	4 (4)	**0**	14.9	137	11
		4.5	**3**	**3**	**5** (5)	**5** (5)	**0**	**0**	51	43
20	12	0	**1**	**1**	**5**	**5**	**0**	**0**	0	0
		1.5	**4**	**4**	**5**	**5**	**0**	**0**	45	20
		3	2	**1**	**5** (5)	**5** (5)	**0**	**0**	98	31
		4.5	**1**	**1**	**5** (5)	**5** (5)	**0**	**0**	8	2
20	24	0	**3**	**3**	**5**	**5**	**0**	**0**	0	0
		1.5	**8**	27	**5**	4	**0**	20	238	35
		3	165	**145**	**5** (2)	4 (3)	**0**	17.3	569	159
		4.5	115	**78**	**5** (5)	**5** (5)	**0**	**0**	252	76
35	12	0	**1**	**1**	**5**	**5**	**0**	**0**	0	0
		1.5	**13**	**13**	**5**	**5**	**0**	**0**	107	58
		3	**16**	25	**5** (2)	**5** (2)	**0**	**0**	156	58
		4.5	**3**	**3**	**5** (5)	**5** (5)	**0**	**0**	24	21
35	24	0	**4**	**4**	**5**	**5**	**0**	**0**	0	0
		1.5	185	**54**	**5**	**5**	**0**	**0**	407	141
		3	**152**	4672	**5** (1)	3	**0**	4.4	1523	261
		4.5	**330**	2140	**5** (5)	4 (4)	**0**	0.7	878	335

Table 7.5: Results for instance files tr40-i-wa100, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. The implementations LG and LGCC are based on the layered graph formulation where LGCC uses connection cuts. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)		solved (worst case)		average gap (in %)		average number of B&B-nodes	
$\|T\|$	B	Γ	LG	LGCC	LG	LGCC	LG	LGCC	LG	LGCC
5	120	0	**12**	16	**5**	**5**	**0**	**0**	0	0
		1.5	**36**	43	**5** (1)	**5** (1)	**0**	**0**	28	4
		3	178	**40**	**5** (3)	**5** (3)	**0**	**0**	81	5
		4.5	28	**23**	**5** (4)	**5** (4)	**0**	**0**	264	35
5	240	0	62	**53**	**5**	**5**	**0**	**0**	0	0
		1.5	213	**177**	**5** (1)	**5** (1)	**0**	**0**	11	1
		3	**261**	516	**5** (4)	**5** (4)	**0**	**0**	364	35
		4.5	259	**251**	**5** (5)	**5** (5)	**0**	**0**	189	61
20	120	0	**18**	**18**	**5**	**5**	**0**	**0**	44	0
		1.5	**1740**	4667	**4**	**4**	**4.8**	14.6	928	84
		3	**4166**	-	**4** (1)	2 (1)	**8.1**	60	953	36
		4.5	1983	**1353**	**4** (3)	3 (2)	**3.3**	40	1592	53
20	240	0	**85**	96	**5**	**5**	**0**	**0**	2	0
		1.5	**5329**	6737	**3**	**3**	**27.1**	34.8	37	1
		3	-	**5799**	1	**3** (1)	65.8	**23.1**	24	13
		4.5	-	-	**1** (1)	**1** (1)	**67**	75.9	10	9
35	120	0	31	**28**	**5**	**5**	**0**	**0**	198	1
		1.5	**9162**	-	**3**	2	**11.4**	60	418	18
		3	-	-	**2**	1	**30.3**	68	715	25
		4.5	-	-	**2** (2)	1 (1)	**47.2**	80	388	14
35	240	0	164	**157**	**5**	**5**	**0**	**0**	1	0
		1.5	-	-	**1**	**1**	**77.6**	78.3	9	2
		3	-	-	**1**	**1**	**78.3**	80	6	11
		4.5	-	-	**1** (1)	0	**78.2**	100	61	2

Table 7.6: Results for instance files te40-i-wa10, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. The implementations LG and LGCC are based on the layered graph formulation where LGCC uses connection cuts. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)		solved (worst case)		average gap (in %)		average number of B&B-nodes	
$\|T\|$	B	Γ	LG	LGCC	LG	LGCC	LG	LGCC	LG	LGCC
5	12	0	7	**2**	**5** (1)	**5** (1)	**0**	**0**	308	1
		1.5	10	**2**	**5** (4)	**5** (4)	**0**	**0**	176	1
		3	4	**2**	**5** (5)	**5** (5)	**0**	**0**	91	2
		4.5	**1**	**1**	**5** (5)	**5** (5)	**0**	**0**	4	0
5	24	0	6157	**33**	3	**5**	6.9	**0**	16415	2
		1.5	5316	**86**	3	**5**	3.2	**0**	34504	7
		3	1546	**57**	3 (2)	**5** (4)	3.6	**0**	34627	6
		4.5	1893	**46**	4 (4)	**5** (5)	2.2	**0**	26764	22
20	12	0	7	**4**	**5**	**5**	**0**	**0**	95	3
		1.5	**30**	39	**5**	**5**	**0**	**0**	418	32
		3	29	**15**	**5** (5)	4 (4)	**0**	2.5	344	42
		4.5	4	**2**	**5** (5)	**5** (5)	**0**	**0**	64	13
20	24	0	1378	**76**	**5**	**5**	**0**	**0**	12120	2
		1.5	-	-	**2**	**2**	**3.6**	14.6	31844	13
		3	-	-	0	0	**10.5**	51.8	24460	25
		4.5	-	-	**1** (1)	0	**8.3**	47.9	27934	47
35	12	0	6	**4**	**5**	**5**	**0**	**0**	227	3
		1.5	**33**	**33**	**5**	4	**0**	1.5	463	48
		3	**47**	60	**5** (3)	**5** (2)	**0**	**0**	472	88
		4.5	7	**4**	**5** (5)	**5** (5)	**0**	**0**	157	48
35	24	0	2744	**488**	3	**5**	2.3	**0**	16306	2
		1.5	-	-	**1**	0	**6.2**	68.9	25157	7
		3	-	-	0	0	**9**	85.9	22650	6
		4.5	-	-	0	0	**6.7**	72.6	26847	2

Table 7.7: Results for instance files te40-i-wa100, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. The implementations LG and LGCC are based on the layered graph formulation where LGCC uses connection cuts. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)		solved (worst case)		average gap (in %)		average number of B&B-nodes	
$\|T\|$	B	Γ	LG	LGCC	LG	LGCC	LG	LGCC	LG	LGCC
5	120	0	2512	**93**	4	**5**	2.9	**0**	2722	2
		1.5	4529	**118**	4 (2)	**5** (3)	0.9	**0**	2586	6
		3	4828	**91**	3 (3)	**5** (5)	6.8	**0**	2914	29
		4.5	1765	**85**	**5** (5)	**5** (5)	**0**	**0**	4368	8
5	240	0	-	**9963**	0	**3** (1)	40.7	**20.5**	4	1
		1.5	-	-	0	**1** (1)	**40.3**	48	24	1
		3	-	**5229**	0	**3** (2)	38.9	**23.9**	15	8
		4.5	-	**2902**	0	**3** (3)	40.6	**26.3**	5	17
20	120	0	-	**605**	2	**4**	3.4	**1.7**	1800	3
		1.5	-	-	0	**1**	**19.8**	59.3	1176	3
		3	-	-	0	**1** (1)	**19.8**	65.2	932	1
		4.5	-	-	0	**1** (1)	**56.9**	80	1250	3
20	240	0	-	-	0	0	**46**	72.5	2	0
		1.5	-	-	0	0	**58.5**	94	2	0
		3	-	-	0	0	**73.3**	100	2	0
		4.5	-	-	0	0	**62.5**	91	3	0
35	120	0	2148	**667**	**5**	**5**	**0**	**0**	1669	2
		1.5	-	-	0	0	**48.2**	100	711	0
		3	-	-	0	0	**57**	100	352	1
		4.5	-	-	0	0	**38**	94.2	594	1
35	240	0	-	-	0	0	**67.6**	92.4	2	0
		1.5	-	-	0	0	**69.5**	100	2	0
		3	-	-	0	0	**71.7**	100	3	0
		4.5	-	-	0	0	**91.3**	100	3	0

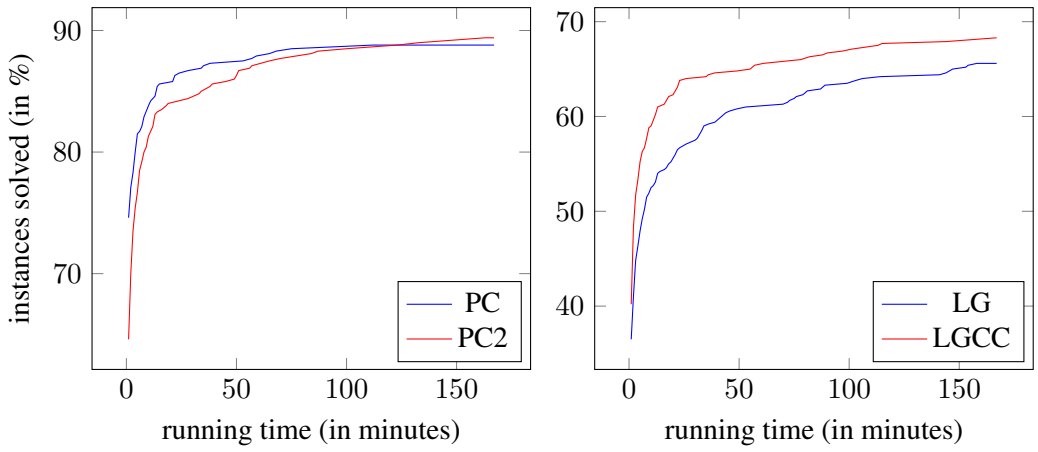


Figure 7.3: Number of solved instances with implementations (a) PC and PC2 and (b) LG and LGCC after a given time. The set of instances contains the instances from the Tables 7.4–7.7.

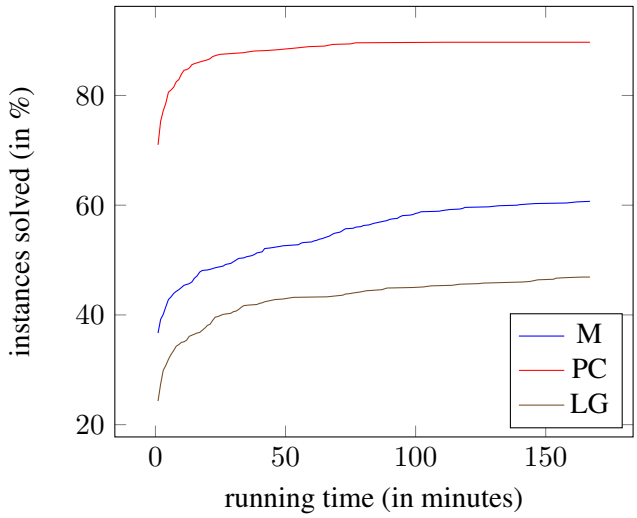


Figure 7.4: Number of solved instances with implementations M, PC and LG after a given time. The set of instances contains the instances from the Tables 7.8–7.13.

Table 7.8: Results for instance files tr40-i-wa10, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. M is based on the MCF formulation, PC on the path-cut formulation and LG on the layered graph formulation. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)			solved (worst case)			average gap (in %)			average number of B&B-nodes		
$\|T\|$	B	Γ	M	PC	LG	M	PC	LG	M	PC	LG	M	PC	LG
5	12	0	2	**1**	**1**	**5**	**5**	**5**	**0**	**0**	**0**	0	850	0
		1.5	4	**1**	**1**	**5** (2)	**5** (2)	**5** (2)	**0**	**0**	**0**	32	442	0
		3	5	**1**	**1**	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	98	1222	19
		4.5	4	**1**	**1**	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	63	1262	0
5	24	0	**1**	**1**	2	**5** (2)	**5** (1)	**5** (2)	**0**	**0**	**0**	0	17	0
		1.5	3	**1**	4	**5** (3)	**5** (3)	**5** (3)	**0**	**0**	**0**	26	568	191
		3	4	**1**	5	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	22	920	137
		4.5	3	**1**	3	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	29	1033	51
20	12	0	54	3	**1**	**5**	**5**	**5**	**0**	**0**	**0**	655	1231	0
		1.5	1798	**1**	4	4	**5**	**5**	2.7	**0**	**0**	2441	1304	45
		3	5438	**1**	2	3 (3)	**5** (5)	**5** (5)	5.6	**0**	**0**	2500	1577	98
		4.5	5222	**1**	**1**	4 (4)	**5** (5)	**5** (5)	3.3	**0**	**0**	2791	1499	8
20	24	0	24	**2**	3	**5**	**5**	**5**	**0**	**0**	**0**	176	1436	0
		1.5	757	**3**	8	**5**	**5**	**5**	**0**	**0**	**0**	983	1823	238
		3	575	**5**	165	**5** (4)	**5** (4)	**5** (2)	**0**	**0**	**0**	1216	5708	569
		4.5	2152	**3**	115	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	1165	3986	252
35	12	0	3691	2	**1**	4	**5**	**5**	2.9	**0**	**0**	4716	1195	0
		1.5	-	**1**	13	0	**5**	**5**	21	**0**	**0**	1261	364	107
		3	-	**2**	16	0	**5** (2)	**5** (2)	19.8	**0**	**0**	1655	435	156
		4.5	-	**1**	3	0	**5** (5)	**5** (5)	19.5	**0**	**0**	2029	800	24
35	24	0	464	5	**4**	4	**5**	**5**	0.8	**0**	**0**	1401	2124	0
		1.5	4782	**4**	185	3	**5**	**5**	18.4	**0**	**0**	943	2117	407
		3	-	**4**	152	0	**5**	**5** (1)	35.3	**0**	**0**	1126	2166	1523
		4.5	-	**8**	330	0	**5** (5)	**5** (5)	40.9	**0**	**0**	952	4528	878

Table 7.9: Results for instance files tr40-i-wa100, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. M is based on the MCF formulation, PC on the path-cut formulation and LG on the layered graph formulation. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)			solved (worst case)			average gap (in %)			average number of B&B-nodes		
$\|T\|$	B	Γ	M	PC	LG	M	PC	LG	M	PC	LG	M	PC	LG
5	120	0	2	**1**	12	**5**	**5**	**5**	**0**	**0**	**0**	4	702	0
		1.5	12	**5**	36	**5** (1)	**5** (1)	**5** (1)	**0**	**0**	**0**	413	1871	28
		3	8	**2**	178	**5** (3)	**5** (3)	**5** (3)	**0**	**0**	**0**	72	1273	81
		4.5	10	**3**	28	**5** (4)	**5** (4)	**5** (4)	**0**	**0**	**0**	53	2077	264
5	240	0	1	**0**	62	**5**	**5**	**5**	**0**	**0**	**0**	0	5	0
		1.5	4	**1**	213	**5** (1)	**5** (1)	**5** (1)	**0**	**0**	**0**	6	54	11
		3	6	**1**	261	**5** (4)	**5** (4)	**5** (4)	**0**	**0**	**0**	235	388	364
		4.5	8	**1**	259	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	21	867	189
20	120	0	73	**4**	18	4	4	**5**	2.2	6.5	**0**	1122	33523	44
		1.5	6584	**5**	1740	**4**	**4**	**4**	**3**	4.4	4.8	3860	40659	928
		3	5890	**4**	4166	**4** (2)	**4** (1)	**4** (1)	3.3	**2.6**	8.1	3712	42899	953
		4.5	5976	**11**	1983	3 (2)	**4** (3)	**4** (3)	3.3	**2.6**	3.3	3839	44270	1592
20	240	0	16	**2**	85	**5**	**5**	**5**	**0**	**0**	**0**	108	1674	2
		1.5	268	**2**	5329	**5**	**5**	3	**0**	**0**	27.1	656	4260	37
		3	242	**4**	-	4 (1)	**5** (1)	1	5	**0**	65.8	1216	38431	24
		4.5	2820	**5**	-	**4** (4)	**4** (4)	1 (1)	3.8	**1.3**	67	2076	42890	10
35	120	0	2372	**3**	31	4	4	**5**	4.7	5.3	**0**	1348	31702	198
		1.5	-	**5**	9162	0	**5**	3	24	**0**	11.4	1273	40685	418
		3	-	**8**	-	0	**5** (1)	2	21.5	**0**	30.3	918	24451	715
		4.5	-	**19**	-	0	**5** (3)	2 (2)	29.6	**0**	47.2	913	7613	388
35	240	0	53	**2**	164	**5**	**5**	**5**	**0**	**0**	**0**	1728	5459	1
		1.5	-	**6**	-	1	**5**	1	80	**0**	77.6	709	37770	9
		3	-	**2**	-	1	**4** (1)	1	32.2	**1.6**	78.3	708	33795	6
		4.5	-	**5**	-	1 (1)	**4** (4)	1 (1)	47.5	**9.4**	78.2	1106	8269	61

Table 7.10: Results for instance files tr40-i-wa1000, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. M is based on the MCF formulation, PC on the path-cut formulation and LG on the layered graph formulation. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)			solved (worst case)			average gap (in %)			average number of B&B-nodes		
$\|T\|$	B	Γ	M	PC	LG	M	PC	LG	M	PC	LG	M	PC	LG
5	1200	0	**2**	**2**	1100	**5**	**5**	**5**	**0**	**0**	**0**	7	2862	0
		1.5	15	**2**	7535	**5** (2)	**5** (2)	3 (1)	**0**	**0**	22.8	154	3360	82
		3	11	**4**	7406	**5** (5)	**5** (5)	3 (3)	**0**	**0**	19.7	72	6655	167
		4.5	8	**4**	3152	**5** (5)	**5** (5)	3 (3)	**0**	**0**	17.8	27	5951	88
5	2400	0	**1**	**1**	-	**5**	**5**	0	**0**	**0**	100	0	1	0
		1.5	7	**2**	-	**5** (2)	**5** (2)	0	**0**	**0**	100	43	879	0
		3	7	**2**	-	**5** (5)	**5** (4)	0	**0**	**0**	100	16	657	0
		4.5	7	**2**	-	**5** (5)	**5** (5)	0	**0**	**0**	100	7	804	0
20	1200	0	1765	**39**	1334	**5**	4	4	**0**	1.2	20	2092	53185	0
		1.5	-	**5**	-	2	**5**	1	5.9	**0**	35.5	3110	9166	50
		3	-	**19**	-	1 (1)	**5** (2)	0	10.3	**0**	97.2	4047	6011	31
		4.5	-	**16**	-	1 (1)	**5** (4)	0	11.8	**0**	87.8	3832	17845	54
20	2400	0	37	**8**	-	**5**	**5**	0	**0**	**0**	100	95	1416	0
		1.5	399	**14**	-	**5** (1)	**5** (1)	0	**0**	**0**	100	1284	6951	0
		3	609	**40**	-	3 (1)	**5** (3)	0	3.9	**0**	100	1572	9834	0
		4.5	6054	**28**	-	3 (3)	**5** (5)	0	5.8	**0**	100	2200	10153	0
35	1200	0	-	**11**	-	1	**5**	1	9.4	**0**	80	2074	39262	0
		1.5	-	**9**	-	1	**5**	1	16.3	**0**	50.1	873	2264	34
		3	-	**8**	-	0	**5**	0	46.5	**0**	100	487	2835	25
		4.5	-	**8**	-	0	**5** (1)	0	69.5	**0**	81.6	473	5854	29
35	2400	0	903	**21**	-	**5**	**5**	0	**0**	**0**	100	1507	3232	0
		1.5	-	**34**	-	1	**5**	0	42.4	**0**	100	740	6122	0
		3	-	**82**	-	1 (1)	**5** (1)	0	41.5	**0**	100	704	17845	0
		4.5	-	**280**	-	1 (1)	**5** (4)	0	45.9	**0**	100	653	41982	0

Table 7.11: Results for instance files te40-i-wa10, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. M is based on the MCF formulation, PC on the path-cut formulation and LG on the layered graph formulation. An explanation of the table entries is given in Section 7.2.

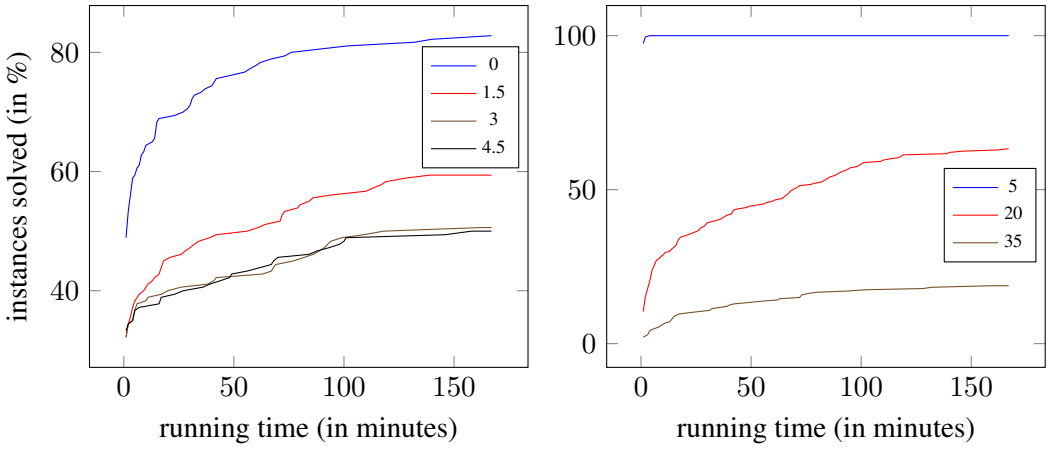
instances			median time (in seconds)			solved (worst case)			average gap (in %)			average number of B&B-nodes		
$\|T\|$	B	Γ	M	PC	LG	M	PC	LG	M	PC	LG	M	PC	LG
5	12	0	**3**	8	7	**5** (1)	**5**	**5** (1)	**0**	**0**	**0**	0	2184	308
		1.5	**3**	**3**	10	**5** (4)	**5** (4)	**5** (4)	**0**	**0**	**0**	9	720	176
		3	4	**2**	4	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	11	1114	91
		4.5	6	2	**1**	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	8	562	4
5	24	0	**2**	3	6157	**5**	**5**	3	**0**	**0**	6.9	0	1943	16415
		1.5	5	**3**	5316	**5** (1)	**5**	3	**0**	**0**	3.2	9	1085	34504
		3	**6**	**6**	1546	**5** (4)	**5** (4)	3 (2)	**0**	**0**	3.6	1	1531	34627
		4.5	**6**	8	1893	**5** (5)	**5** (5)	4 (4)	**0**	**0**	2.2	14	1908	26764
20	12	0	1589	93	**7**	**5**	**5**	**5**	**0**	**0**	**0**	715	14398	95
		1.5	-	**6**	30	2	**5**	**5**	6.6	**0**	**0**	1033	2676	418
		3	-	**7**	29	0	**5** (5)	**5** (5)	12.6	**0**	**0**	1234	3997	344
		4.5	-	9	**4**	2 (2)	**5** (5)	**5** (5)	8.8	**0**	**0**	1445	3728	64
20	24	0	75	**35**	1378	**5**	**5**	**5**	**0**	**0**	**0**	25	18540	12120
		1.5	4211	**433**	-	4	**5**	2	1.9	**0**	3.6	399	53165	31844
		3	-	**363**	-	2 (1)	**4** (1)	0	22.7	**0.7**	10.5	599	53050	24460
		4.5	-	**565**	-	1 (1)	**5** (3)	1 (1)	6.7	**0**	8.3	556	88920	27934
35	12	0	-	83	**6**	0	4	**5**	6	0.7	**0**	1112	29023	227
		1.5	-	**8**	33	0	**5**	**5**	65.7	**0**	**0**	290	1826	463
		3	-	**10**	47	0	**5** (3)	**5** (3)	51.5	**0**	**0**	373	1488	472
		4.5	-	13	**7**	0	**5** (5)	**5** (5)	33.8	**0**	**0**	303	2557	157
35	24	0	-	-	**2744**	2	0	**3**	**1.2**	5.2	2.3	891	116764	16306
		1.5	-	-	-	0	**1**	**1**	45.9	**4.5**	6.2	76	88511	25157
		3	-	-	-	0	**1**	0	65.8	**5.5**	9	47	109205	22650
		4.5	-	-	-	0	0	0	83.5	**5.1**	6.7	57	118491	26847

Table 7.12: Results for instance files te40-i-wa100, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. M is based on the MCF formulation, PC on the path-cut formulation and LG on the layered graph formulation. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)			solved (worst case)			average gap (in %)			average number of B&B-nodes		
$\|T\|$	B	Γ	M	PC	LG	M	PC	LG	M	PC	LG	M	PC	LG
5	120	0	**3**	6	2512	**5**	**5**	4	**0**	**0**	2.9	0	2019	2722
		1.5	8	**4**	4529	**5** (3)	**5** (3)	4 (2)	**0**	**0**	0.9	4	1200	2586
		3	7	**6**	4828	**5** (5)	**5** (5)	3 (3)	**0**	**0**	6.8	3	1259	2914
		4.5	6	**4**	1765	**5** (5)	**5** (5)	**5** (5)	**0**	**0**	**0**	1	1260	4368
5	240	0	**2**	3	-	**5** (2)	**5** (1)	0	**0**	**0**	40.7	0	1905	4
		1.5	4	**2**	-	**5** (3)	**5** (3)	0	**0**	**0**	40.3	0	1998	24
		3	**4**	**4**	-	**5** (3)	**5** (3)	0	**0**	**0**	38.9	4	1756	15
		4.5	10	**4**	-	**5** (5)	**5** (5)	0	**0**	**0**	40.6	12	1552	5
20	120	0	3561	**455**	-	**4**	3	2	**0.8**	2	3.4	1105	58234	1800
		1.5	-	**103**	-	1	**5**	0	7.3	**0**	19.8	756	20832	1176
		3	-	**218**	-	1 (1)	**5** (3)	0	14.6	**0**	19.8	774	41037	932
		4.5	-	**350**	-	1 (1)	**4** (4)	0	13.8	**0.8**	56.9	549	58877	1250
20	240	0	92	**54**	-	**5**	**5**	0	**0**	**0**	46	237	43793	2
		1.5	1035	**187**	-	**5**	4	0	**0**	0.5	58.5	359	42991	2
		3	-	**216**	-	2	**4**	0	13.6	**0.7**	73.3	362	40141	2
		4.5	-	**434**	-	2 (1)	**4** (3)	0	41.1	**0.3**	62.5	404	57739	3
35	120	0	-	**1269**	2148	0	3	**5**	6.3	1.9	**0**	750	78031	1669
		1.5	-	**190**	-	0	**4**	0	100	**1.1**	48.2	123	47198	711
		3	-	**46**	-	0	**5** (2)	0	83	**0**	57	91	35528	352
		4.5	-	**130**	-	0	**4** (3)	0	82.3	**2.1**	38	122	46149	594
35	240	0	**4323**	-	-	**4**	0	0	**0.3**	3.4	67.6	517	116320	2
		1.5	-	-	-	0	**1**	0	44.3	**4.4**	69.5	110	89423	2
		3	-	-	-	0	**2**	0	80.4	**2.7**	71.7	75	112980	3
		4.5	-	-	-	0	**1**	0	80.6	**3.3**	91.3	45	110857	3

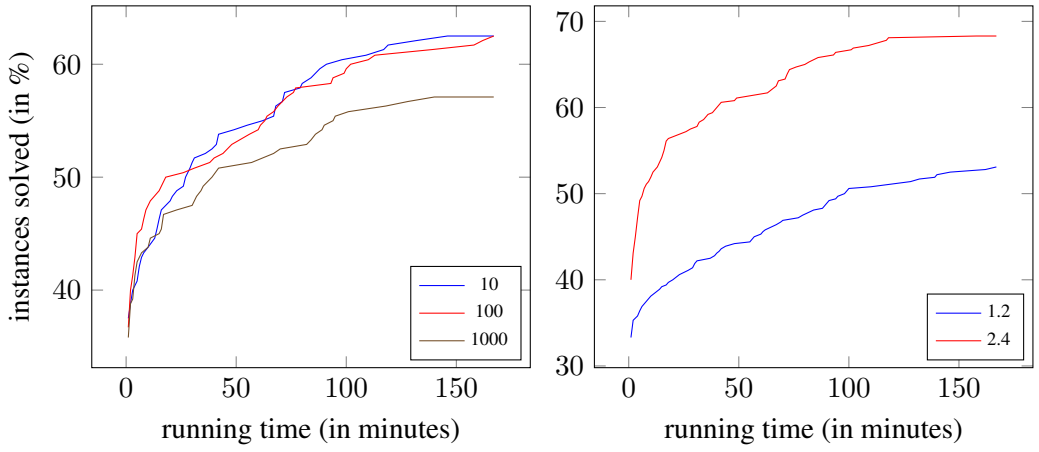
Table 7.13: Results for instance files te40-i-wa1000, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-RRDCST problem. M is based on the MCF formulation, PC on the path-cut formulation and LG on the layered graph formulation. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)			solved (worst case)			average gap (in %)			average number of B&B-nodes		
$\|T\|$	B	Γ	M	PC	LG	M	PC	LG	M	PC	LG	M	PC	LG
5	1200	0	**3**	12	-	**5**	**5**	1	**0**	**0**	40.4	3	2775	83
		1.5	8	**3**	-	**5** (2)	**5** (2)	0	**0**	**0**	44.3	47	891	95
		3	9	**8**	-	**5** (5)	**5** (5)	0	**0**	**0**	35.9	129	2313	158
		4.5	7	**6**	-	**5** (5)	**5** (5)	0	**0**	**0**	36.8	161	2682	197
5	2400	0	**2**	3	-	**5** (1)	**5**	0	**0**	**0**	100	0	1760	0
		1.5	5	**3**	-	**5** (2)	**5** (2)	0	**0**	**0**	100	0	1658	0
		3	5	**3**	-	**5** (4)	**5** (4)	0	**0**	**0**	100	2	1878	0
		4.5	**5**	7	-	**5** (5)	**5** (5)	0	**0**	**0**	100	2	1782	0
20	1200	0	3414	**548**	-	**4**	**4**	1	0.5	**0.5**	55.8	1380	56514	13
		1.5	-	**130**	-	1	**5**	0	24.8	**0**	100	924	24306	4
		3	-	**72**	-	0	**5** (1)	0	15.1	**0**	84.7	1014	37500	12
		4.5	-	**258**	-	0	**5** (4)	0	11.8	**0**	83.2	888	49975	19
20	2400	0	151	**61**	-	**5**	**5**	0	**0**	**0**	100	209	25164	0
		1.5	-	**241**	-	2	**5**	0	22.1	**0**	100	583	44397	0
		3	7063	**113**	-	3	**4**	0	4.4	**0.7**	100	793	48761	0
		4.5	-	**69**	-	1	**3** (1)	0	5.9	**1.4**	100	729	59805	0
35	1200	0	-	-	-	0	**2**	0	8.6	**3.7**	100	956	80796	0
		1.5	-	**111**	-	0	**5**	0	63.5	**0**	100	150	52912	0
		3	-	**1176**	-	0	**4**	0	100	**0.8**	100	125	63376	0
		4.5	-	**391**	-	0	**4**	0	100	**0.9**	100	158	61995	0
35	2400	0	**570**	-	-	**3**	2	0	**0.8**	3.2	100	362	83029	0
		1.5	-	**593**	-	1	**3**	0	26.4	**2.7**	100	187	67617	0
		3	-	**608**	-	1	**3**	0	43.8	**4.1**	100	255	54458	0
		4.5	-	-	-	1	**2**	0	62.5	**5.6**	100	191	58204	0



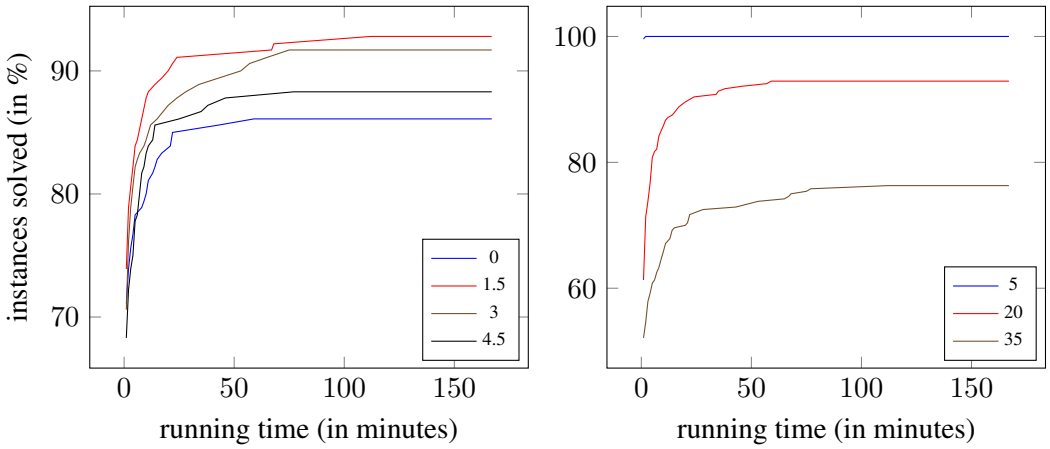
(a) (b)

Figure 7.5: Number of solved instances with implementation M after a given time. The set of instances contains the instances from the Tables 7.8–7.13. The different values of Γ are compared in (a) and the different values of $|T|$ are compared in (b).



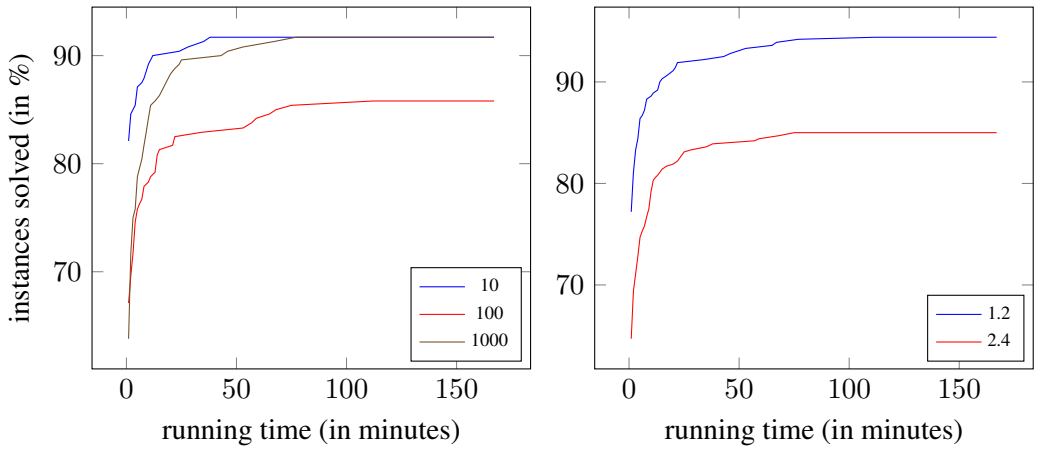
(a) (b)

Figure 7.6: Number of solved instances with implementation M after a given time. The set of instances contains the instances from the Tables 7.8–7.13. The different values of $d_{\max}$ are compared in (a) and the different values of $\frac{B}{d_{\max}}$ are compared in (b).



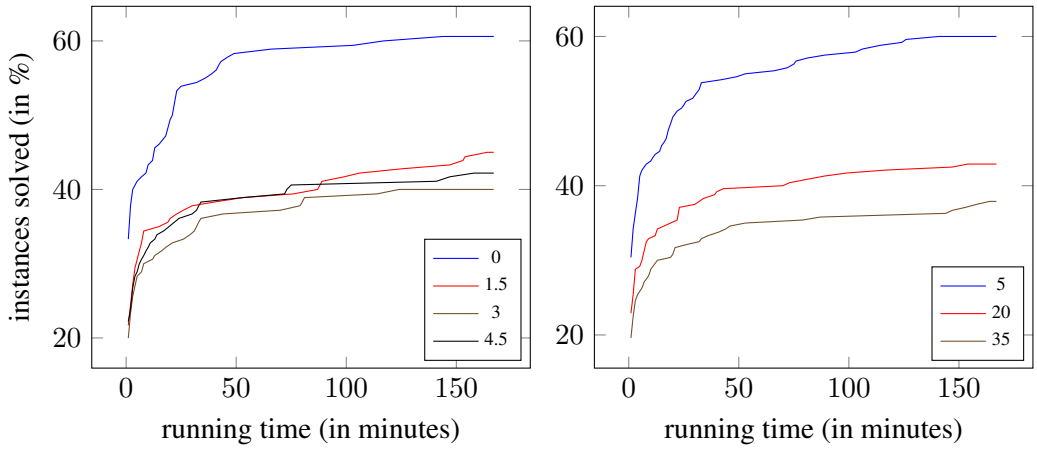
(a) (b)

Figure 7.7: Number of solved instances with implementation PC after a given time. The set of instances contains the instances from the Tables 7.8–7.13. The different values of Γ are compared in (a) and the different values of $|T|$ are compared in (b).



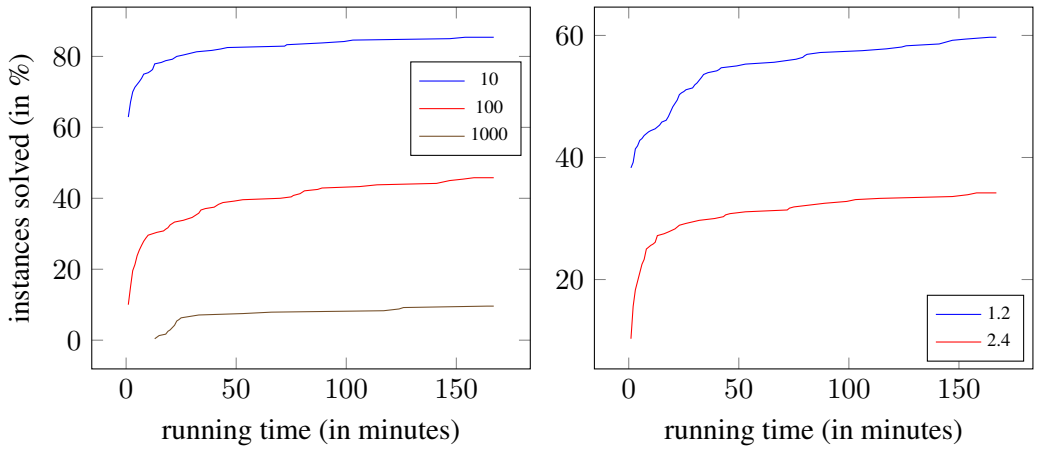
(a) (b)

Figure 7.8: Number of solved instances with implementation PC after a given time. The set of instances contains the instances from the Tables 7.8–7.13. The different values of $d_{\max}$ are compared in (a) and the different values of $\frac{B}{d_{\max}}$ are compared in (b).



(a) (b)

Figure 7.9: Number of solved instances with implementation LG after a given time. The set of instances contains the instances from the Tables 7.8–7.13. The different values of Γ are compared in (a) and the different values of $|T|$ are compared in (b).



(a) (b)

Figure 7.10: Number of solved instances with implementation LG after a given time. The set of instances contains the instances from the Tables 7.8–7.13. The different values of $d_{\max}$ are compared in (a) and the different values of $\frac{B}{d_{\max}}$ are compared in (b).

Table 7.14: Results for instance files te40-i-wa10, $i \in \{1, 2, 3, 4, 5\}$ of the 0.5-SRDCST problem. PC is based on the path-cut formulation and LG on the layered graph formulation. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)		solved instances		average gap (in %)		average number of B&B-nodes	
$\|T\|$	B	u_p	PC	LG	PC	LG	PC	LG	PC	LG
5	12	-1	**9**	510	**5**	**5**	**0**	**0**	1972	16894
		0	8	**7**	**5**	**5**	**0**	**0**	2184	308
		1	4	**3**	**5**	**5**	**0**	**0**	534	27
		2	**1**	2	**5**	**5**	**0**	**0**	216	18
5	24	-1	**3**	-	**5**	1	**0**	19.5	1310	21414
		0	**3**	5138	**5**	3	**0**	6.9	1943	16142
		1	**4**	2001	**5**	4	**0**	2.9	1192	19768
		2	**7**	1728	**5**	**5**	**0**	**0**	3189	23092
20	12	-1	**878**	6375	**4**	**4**	0.8	**0.3**	74595	41660
		0	92	**7**	**5**	**5**	**0**	**0**	14398	95
		1	**4**	7	**5**	**5**	**0**	**0**	2191	58
		2	**6**	13	**5**	**5**	**0**	**0**	1078	162
20	24	-1	**33**	-	**5**	0	**0**	9.1	7899	15012
		0	**34**	1617	**5**	**5**	**0**	**0**	18540	12120
		1	**610**	4948	4	**5**	0.7	**0**	69962	34310
		2	**170**	5418	**5**	4	**0**	0.8	43277	35846
35	12	-1	-	-	0	**1**	5.2	**3.7**	110343	32130
		0	74	**5**	4	**5**	0.7	**0**	29025	227
		1	**4**	15	**5**	**5**	**0**	**0**	2229	170
		2	**2**	10	**5**	**5**	**0**	**0**	497	72
35	24	-1	-	-	0	0	**5.6**	10.7	108614	11365
		0	-	**2875**	0	**3**	5.2	**2.3**	116779	16186
		1	-	-	1	**2**	5.5	**2.8**	107613	29432
		2	-	-	**2**	**2**	**2.7**	2.7	80794	33002

Table 7.15: Results for instance files of the 0.5-RRDCST problem from [44] with $|V| = 100$. M is based on the MCF formulation, PC on the path-cut formulation, LG on the layered graph formulation and LGCC on the layered graph formulation with connection cuts. An explanation of the table entries is given in Section 7.2.

instances			median time (in seconds)				solved (worst case)				average gap (in %)			
$\|T\|$	B	Γ	M	PC	LG	LGCC	M	PC	LG	LGCC	M	PC	LG	LGCC
15	25	0	1595	51	12	**10**	28	29	**30**	**30**	0.3	1.6	**0**	**0**
		1.5	-	**16**	24	40	12	**30**	**30**	26	8.3	**0**	**0**	5
		3	-	**24**	49	150	7 (1)	**30** (2)	**30** (2)	26 (1)	13	**0**	**0**	7.6
		4.5	-	**42**	51	156	6 (3)	28 (16)	**30** (16)	26 (14)	19.8	1.1	**0**	10.5
15	40	0	-	511	19	**18**	15	16	**30**	**30**	4.8	20.8	**0**	**0**
		1.5	-	**113**	531	-	1	22	**29**	15	39.5	6.7	**0.4**	45.8
		3	-	**177**	1039	-	1	20 (3)	**26** (2)	8 (1)	44	9.2	**2.4**	65.9
		4.5	-	**390**	1383	-	1 (1)	20 (13)	**26** (15)	10 (5)	46.3	11.3	**1**	60.9
50	25	0	-	147	**11**	**11**	0	25	**30**	**30**	80.9	1.9	**0**	**0**
		1.5	-	**14**	216	646	0	**30**	**30**	26	100	**0**	**0**	10.2
		3	-	**35**	550	-	0	27	**28**	13	100	0.8	**0.2**	16.3
		4.5	-	**130**	1419	-	0	21 (2)	**26** (2)	10 (1)	100	4.6	**1**	27.6
50	40	0	-	-	**25**	**25**	0	12	**30**	**30**	89.5	17	**0**	**0**
		1.5	-	**421**	-	-	0	**19**	14	5	100	**5.7**	6.3	68.8
		3	-	-	-	-	0	**13**	5	0	100	**11**	33.3	94.4
		4.5	-	-	-	-	0	**8**	2	0	100	**22.5**	40.2	94.6
85	25	0	-	44	**12**	**12**	0	29	**30**	**30**	100	0.2	**0**	**0**
		1.5	-	**14**	203	486	0	**30**	**30**	24	100	**0**	**0**	1.1
		3	-	**30**	600	-	0	**29**	27	12	100	**0.2**	0.3	16.3
		4.5	-	**90**	1900	-	0	**24** (2)	**24** (2)	5 (1)	100	**0.9**	1.1	28.8
85	40	0	-	-	**29**	31	0	14	**30**	**30**	100	7.2	**0**	**0**
		1.5	-	**226**	-	-	0	**22**	8	1	100	**1.8**	12.9	81.3
		3	-	**927**	-	-	0	**18**	2	1	100	**3.5**	36.2	91
		4.5	-	-	-	-	0	**7**	2	0	100	**8.3**	44.7	100

CHAPTER 8

Conclusions

This thesis is about robust optimization and stochastic programming and their application to the rooted delay-constrained Steiner tree problem with uncertain delays. For each of them, one approach is chosen which results in a robust and a stochastic version of the problem. The stochastic version has the advantage that it is built upon a solid stochastic foundation such that the definition of optimality is based on important probabilistic properties. This ensures a high quality of the solutions. Some formulations as integer linear program are presented. The multi commodity flow formulation uses only polynomially many variables and constraints. One of its limitations is that it cannot be applied to different robust or stochastic approaches analogously, because it depends on the problem specific delay constraints. In our case the delay constraints of the robust approach are suited for this formulation after a linearisation. The path-cut formulation uses exponentially many constraints but can be applied to different approaches more easily, since the delay constraints are not encountered explicitly. Feasible paths can be determined separately in a subroutine. In comparison with the other developed algorithms, the implementation based on this formulation has the best performance for most instances. Quite recently, layered graph formulations gained popularity in algorithms for a specific class of tree problems. Extending this concept to the robust and the stochastic version of the problem proves to be difficult. Moreover, the implementation based on the layered graph formulation has an increased running time for uncertain delays. But there is still a set of instances where this implementation outperforms the others. An efficient algorithm based on the path formulation may be able to solve many instances of the robust and the stochastic problem even faster as with the other formulations, but its construction could be a bit more difficult.

Some different strategies are presented to reduce the size of the problem instances. This can be achieved by deleting selected edges of the instance graph. Sometimes instance transformations can be used as an alternative way of introducing robustness. This leads to fast algorithms, but the quality of the solutions can be rather poor because of an undesired definition of optimality.

With the improvements on ILP solvers it will be possible to solve the robust and the stochastic problem faster in future. Certainly, there are also ways to improve the performance without

changing the solver. However, all discussed problems are $\mathcal{NP}$-hard which means that there is always an instance of moderate size too hard to solve to optimality. As today's networks grow larger and larger, it may be a too strong limitation if there are no solutions for such instances. There are a lot of heuristic approaches to solve the deterministic problem. It may be fruitful to study heuristic algorithms also in combination with the concept of uncertainty.

Bibliography

[1] M. Aissa and A. B. Mnaouer. A new delay-constrained algorithm for multicast routing tree construction. *International Journal of Communication Systems*, 17(10):985–1000, 2004.

[2] E. Álvarez Miranda, I. Ljubić, and P. Toth. Exact approaches for solving robust prize-collecting Steiner tree problems. *European Journal of Operational Research*, 229(3):599 – 612, 2013.

[3] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13, 1999.

[4] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424, 2000.

[5] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1-3):49–71, 2003.

[6] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[7] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, 1997.

[8] I. Bomze, M. Chimani, M. Jünger, I. Ljubić, P. Mutzel, and B. Zey. Solving two-stage stochastic Steiner tree problems by two-stage branch-and-cut. In O. Cheong, K.-Y. Chwa, and K. Park, editors, *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 427–439. Springer Berlin Heidelberg, 2010.

[9] C. Büsing and F. D'Andreagiovanni. New results about multi-band uncertainty in robust optimization. In *Proceedings of the 11th International Conference on Experimental Algorithms*, SEA'12, pages 63–74. Springer Berlin Heidelberg, 2012.

[10] G. Calafiore and M.C. Campi. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming*, 102(1):25–46, 2005.

[11] V. Chankong and Y. Y. Haimes. *Multiobjective decision making : theory and methodology*. New York : North Holland, 1983.

[12] A. Charnes and W. W. Cooper. Chance-constrained programming. *Management Science*, 6(1):73–79, 1959.

[13] X. Chen, M. Sim, and P. Sun. A robust optimization perspective on stochastic programming. *Operations Research*, 55(6):1058–1071, 2007.

[14] B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[15] G. Dahl, L. Gouveia, and C. Requejo. On formulations and methods for the hop-constrained minimum spanning tree problem. In M.G.C. Resende and P.M. Pardalos, editors, *Handbook of Optimization in Telecommunications*, pages 493–515. Springer US, 2006.

[16] G. B. Dantzig. Linear programming under uncertainty. *Management Science*, 1(3-4):197–206, 1955.

[17] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[18] D.-Z. Du, Y. Zhang, and Q. Feng. On better heuristic for Euclidean Steiner minimum trees. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 431–439, 1991.

[19] L. El Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, 1997.

[20] L. El Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, 9(1):33–52, 1998.

[21] E. Erdoğan and G. Iyengar. Ambiguous chance constrained problems and robust optimization. *Mathematical Programming*, 107(1):37–61, 2004.

[22] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[23] C. E. Gounaris, W. Wiesemann, and C. A. Floudas. The robust capacitated vehicle routing problem under demand uncertainty. *Operations Research*, 61(3):677–693, 2013.

[24] L. Gouveia. Using variable redefinition for computing lower bounds for minimum spanning and Steiner trees with hop constraints. *INFORMS Journal on Computing*, 10(2):180–188, 1998.

[25] L. Gouveia, A. Paias, and D. Sharma. Modeling and solving the rooted distance-constrained minimum spanning tree problem. *Computers & Operations Research*, 35(2):600–613, 2008.

[26] L. Gouveia, L. Simonetti, and E. Uchoa. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. *Mathematical Programming*, 128(1-2):123–148, 2011.

[27] A. Gupta and M. Pál. Stochastic Steiner trees without a root. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming*, ICALP'05, pages 1051–1063. Springer-Verlag, 2005.

[28] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[29] H. Ishii and T. Nishida. Stochastic bottleneck spanning tree problem. *Networks*, 13(3):443–449, 1983.

[30] H. Ishii, S. Shiode, T. Nishida, and Y. Namasuya. Stochastic spanning tree problem. *Discrete Applied Mathematics*, 3(4):263 – 273, 1981.

[31] P. Kall and S. W. Wallace. Stochastic programming. *Operations Research*, 4(1):1–2, 2003.

[32] O. Klopfenstein and D. Nace. A robust approach to the chance-constrained knapsack problem. *Operations Research Letters*, 36(5):628 – 632, 2008.

[33] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicasting for multimedia applications. In *Proceedings of the 11th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM)*, volume 3, pages 2078–2085. IEEE, 1992.

[34] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, 1993.

[35] V. Leggieri, M. Haouari, and C. Triki. The Steiner tree problem with delays: A compact formulation and reduction procedures. *Discrete Applied Mathematics*, 2011.

[36] M. Leitner, M. Ruthmair, and G. R. Raidl. Stabilized branch-and-price for the rooted delay-constrained Steiner tree problem. In J. Pahl, T. Reiners, and S. Voß, editors, *Network Optimization*, volume 6701 of *Lecture Notes in Computer Science*, pages 124–138. Springer Berlin Heidelberg, 2011.

[37] M. Leitner, M. Ruthmair, and G. R. Raidl. Stabilizing branch-and-price for constrained tree problems. *Networks*, 61(2):150–170, 2013.

[38] B. Liu. Uncertainty theory. In *Uncertainty Theory*, volume 154 of *Studies in Fuzziness and Soft Computing*, pages 205–234. Springer Berlin Heidelberg, 2007.

[39] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[40] M. Monaci, U. Pferschy, and P. Serafini. Exact solution of the robust knapsack problem. *Computers & Operations Research*, 40(11):2625 – 2631, 2013.

[41] R. Montemanni, J. Barta, M. Mastrolilli, and L. M. Gambardella. The robust traveling salesman problem with interval data. *Transportation Science*, 41(3):366–381, 2007.

[42] J. M. Mulvey, R. J. Vanderbei, and S. A. Zenios. Robust optimization of large-scale systems. *Operations Research*, 43(2):264–281, 1995.

[43] A. Nemirovski and A. Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2007.

[44] M. Ruthmair. *On Solving Constrained Tree Problems and an Adaptive Layers Framework*. PhD thesis, Vienna University of Technology, 2012.

[45] M. Ruthmair and G. R. Raidl. A layered graph model and an adaptive layers framework to solve delay-constrained minimum tree problems. In O. Günlük and G. Woeginger, editors, *Integer Programming and Combinatoral Optimization*, volume 6655 of *Lecture Notes in Computer Science*, pages 376–388. Springer Berlin Heidelberg, 2011.

[46] H.F. Salama, D.S. Reeves, and Y. Viniotis. The delay-constrained minimum spanning tree problem. In *Proceedings of the 2nd IEEE Symposium on Computers and Communications*, pages 699–703. IEEE Computer Society, 1997.

[47] A. Schrijver. *Theory of linear and integer programming*. Wiley, 1998.

[48] M. Sim. *Robust Optimization*. PhD thesis, Massachusetts Institute of Technology, 2004.

[49] N. Skorin-Kapov and M. Kos. The application of Steiner trees to delay constrained multicast routing: a tabu search approach. In *Proceedings of the 7th International Conference on Telecommunications*, volume 2, pages 443–448, 2003.

[50] A. L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

[51] A. Thiele. Robust stochastic programming with uncertain probabilities. *IMA Journal of Management Mathematics*, 19(3):289–321, 2007.

[52] R. Viertl. *Einführung in die Stochastik - mit Elementen der Bayes-Statistik und der Analyse unscharfer Information*. Springer Wien New York, 2003.

[53] L. A. Wolsey. *Integer programming*. Wiley, 1998.

[54] H. Yaman, O. E. Karaşan, and M. Ç. Pinar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29(1):31 – 40, 2001.

[55] G. Yu and J. Yang. On the robust shortest path problem. *Computers and Operations Research*, 25(6):457–468, 1998.

[56] Q. Zhang and Y.-W. Leung. An orthogonal genetic algorithm for multimedia multicast routing. *IEEE Transactions on Evolutionary Computation*, 3(1):53–62, 1999.