Master Arbeit im Studiengang Informatik

Optimal Virtualized In-Network Processing
with Applications to Aggregation and Multicast

Matthias Johannes Rost
mrost@inet.tu-berlin.de
Matrikelnummer: 338304

21. Januar 2014

Gutachter

Prof. Anja Feldmann, Ph.D., Technische Universität Berlin
Prof. Dr. Andreas Bley, Universität Kassel

Betreuer

Dr. Stefan Schmid, Technische Universität Berlin

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 21. Januar 2014

_____

Unterschrift

Abstract

Virtualization has become a crucial ingredient in efficiently utilizing and sharing networks. New concepts as Networks Functions Virtualization envision the placement of computational resources, known as universal nodes, throughout ISP networks, such that services can be deployed flexibly on top of these nodes. In data centers such computational ressources are readily available at middleboxes or can be spawned as Virtual Machines.

This thesis considers how the communication services of multicast and aggregation can be *optimally* deployed in virtualized environments. As both these communication schemes rely heavily on in-network processing to e.g. filter, aggregate or duplicate data, the question arises *where* to place in-network processing functionality and *how* to route data. As installing processing functionality either comes at monetary costs (e.g. in ISP networks) or at opportunistic costs due to limited computational resources (e.g. in data centers), we consider the problem of *jointly* minimizing the number of processing locations and the bandwidth.

Interestingly, the corresponding combinatorial optimization problem has not been studied before and is introduced in this thesis as the Constrained Virtual Steiner Arborescence Problem (CVSAP). Despite the novelty of the problem, this thesis provides a comprehensive study of CVSAP comprising results of both theoretical and practical significance:

1. The computational complexity of CVSAP and weaker variants is studied and the inapproximability of CVSAP (unless $P = NP$ holds) is established and three approximation algorithms for variants are obtained.

2. Based on the computational hardness of CVSAP, exact algorithms are studied, yielding a multi-commodity and a single-commodity flow Integer Programming formulation. Together with a *novel* flow decomposition scheme the single-flow formulation yields the VirtuCast algorithm which represents the main theoretical contribution of this thesis.

3. While the inapproximability result for CVSAP does not allow for deriving approximation algorithms, several linear heuristics and a purely combinatorial heuristic are developed.

4. Bridging theoretical and practical considerations, all presented algorithms for CVSAP are implemented and studied in an extensive computational evaluation on different network topologies. The results show that the VirtuCast algorithm - paired with the developed heuristics - can be applied to solve realistically sized instances with more than thousand nodes to within 6% of optimality.

Zusammenfassung

Virtualisierung spielt eine zentrale Rolle, um Ressourcen in Netzwerken effizient zu benutzen und zu teilen. Neuartige Konzepte wie zum Beispiel 'Networks Functions Virtualization' sehen vor, universelle Rechenknoten in den Netzwerken von Internet Service Providern zu platzieren, um Dienste flexibel unter Verwendung dieser bereitstellen zu können. Schon heutzutage existieren in Rechenzentren solche Rechenkapazitäten in der Form von 'Middleboxes' bzw. können als virtuelle Maschinen erzeugt werden.

Im Lichte dieser Entwicklung befasst sich diese Masterarbeit mit der Frage wie Mehrpunktverbindungsdienste und Aggregationsdienste in virtualisierten Netzwerken *optimal* bereitgestellt werden können. Sowohl Mehrpunkt verbindungsdienste wie auch Aggregations- dienste basieren auf der Fähigkeit innerhalb des Netzwerks Datenströme zu filtern, zu aggregieren und zu duplizieren. Dies wirft die Frage auf, auf welchen Netzwerkknoten die Funktionalität zur Verarbeitung von Datenströmen platziert werden soll und entlang welcher Wege Daten zwischen diesen ausgetauscht werden sollen. Da die Installation von Datenstromverarbeitungsfähigkeiten auf Netzwerkknoten entweder monetäre Kosten oder opportune Kosten, z.B. durch die Verwendung von begrenzten Rechenkapazitäten, nach sich zieht, wird in dieser Arbeit die gleichzeitige Minimierung der Anzahl von Netzwerkknoten, welche Datenströme verarbeiten, und der Verwendung von Bandbreite zum Kommunikationsaustausch angestrebt.

Interessanterweise wurde dieses Problem im Rahmen der kombinatorischen Optimierung noch nicht untersucht und wird folglicherweise in dieser Arbeit als 'Constrained Virtual Steiner Arborescence Problem' (CVSAP) eingeführt. Entgegen der Tatsache, dass diese Masterarbeit eben dieses Problem zum ersten Mal definiert, enthält diese Arbeit eine umfassende Untersuchung von CVSAP, mit den folgenden praxisrelevanten als auch theoretischen Resultaten:

1. Die zugrundeliegende Komplexität von CVSAP und abgeschwächten Varianten wird untersucht und die Nichtapproximierbarkeit von CVSAP, sofern nicht $P = NP$ gilt, bewiesen. Weiterhin werden Approximationsalgorithmen für drei abgeschwächte Varianten aufgezeigt.

2. Bedingt durch die Komplexität von CVSAP werden im Rahmen dieser Arbeit exakte Lösungsansätze unter Verwendung von ganzzahlig linearer Optimierung entwickelt. Insbesondere wird CVSAP sowohl als ein Mehrgüter-Flussproblem, sowie auch als ein Eingüter-Flussproblem formuliert. Zusammen mit einem neuartigen Flusszerlegungsalgorithmus wird die Eingüter-Fluss-Formulierung verwandt, um den VirtuCast Algorithmus herzuleiten. Dieser stellt das theoretische Herzstück dieser Arbeit dar.

3. Bedingt durch das Nichtapproximierbarkeitsresultat für CVSAP, werden anstatt von Approximationsalgorithmen mehrere lineare wie auch kombinatorische Heuristiken entwickelt.

4. Alle im Rahmen dieser Arbeit entwickelten Algorithmen wurden implementiert und werden einer umfassenden Untersuchung auf verschiedenen Topologien unterzogen. Die Auswertung dieser Untersuchung zeigt, dass der VirtuCast Algorithmus - zusammen mit den entwickelten Heuristiken - genutzt werden kann, um CVSAP auf realistischen Problemgrößen mit mehreren tausend Knoten bis auf einen Faktor von 6% optimal lösen zu können.

Contents

1. Introduction **1**
- 1.1. Model Outline . . . 3
- 1.2. Applications . . . 4
- 1.3. Overview . . . 7
- 1.4. Contribution . . . 7

I. Theory of CVSAP **9**

2. The Constrained Virtual Steiner Arborescence Problem ✶ **10**
- 2.1. Notation . . . 10
- 2.2. Definition of the Constrained Virtual Steiner Arborescence Problem . . . 11
- 2.3. Inapproximability of CVSAP . . . 14
- 2.4. Variants of CVSAP . . . 14

3. Approximation of CVSAP Variants **18**
- 3.1. 8-Approximation of VSTP via CFLP . . . 19
- 3.2. Equivalence of VSAP and SAP . . . 23
- 3.3. Approximation of NVSTP via DNSTP . . . 27

II. Exact Algorithms for CVSAP **33**

4. A Multi-Commodity Flow Formulation ✶ **34**
- 4.1. Notation . . . 34
- 4.2. The MIP Model . . . 34
- 4.3. Implementation . . . 36

5. VirtuCast Algorithm **37**
- 5.1. The IP Model . . . 37
- 5.2. Flow Decomposition . . . 39
- 5.3. Runtime Analysis for Decompose . . . 45
- 5.4. Implementation . . . 46

III. Heuristics for CVSAP 49

6. Overview and Common Algorithms 50
6.1. Employed Known Algorithms and Definitions . . . 50
6.2. Local Search Procedure PruneSteinerNodes ✶ . . . 53

7. Combinatorial Heuristic GreedySelect 55
7.1. Synopsis of Algorithm GreedySelect . . . 55
7.2. Runtime of Algorithm GreedySelect . . . 57

8. LP-Based Heuristics 59
8.1. Heuristic FlowDecoRound ✶ . . . 59
8.2. Algorithm PartialDecompose . . . 62
8.3. Algorithm Virtual Capacitated Prim Connect . . . 63
8.4. Abstract Interface to LP Solver . . . 65
8.5. Greedy Diving Heuristics . . . 66
8.6. Multiple Shots Heuristics . . . 69
8.7. Runtime Considerations . . . 71

IV. Computational Evaluation 72

9. Outline of the Computational Evaluation 73
9.1. Notation & Measures . . . 74
9.2. General Computational Setup . . . 74

10. Topologies 75
10.1. Selected Topologies . . . 75
10.2. Generation Parameters . . . 76
10.3. Fat Tree . . . 77
10.4. 3D Torus . . . 77
10.5. IGen . . . 78

11. Separation & Branching Parameters 79
11.1. Considered Parameters . . . 79
11.2. General Methodoloy . . . 80
11.3. Initial Parameter Validation . . . 80
11.4. Final Parameter Validation . . . 88
11.5. Final Separation & Branching Parameters . . . 90

12. Performance of LP-Based Heuristics 100
12.1. Methodology . . . 100
12.2. Computational Setup . . . 101
12.3. Overview of Results . . . 102
12.4. Detailed Topology-Dependent Analysis . . . 104

13. Performance of the Final VirtuCast Solver 117

14. Evaluation of alternative Algorithms 119
14.1. GreedySelect 119
14.2. Performance of Formulation A-CVSAP-MCF 120
14.3. VirtuCast with SCIP's Heuristics 120

V. Conclusion 125

15. Related Work 126
15.1. Applicability of CVSAP 126
15.2. Related Theoretical Problems 127
15.3. Integer Programming Formulations 127

16. Summary of Results & Future Work 128

Bibliography 129

1. Introduction

Virtualization is one of the key enablers for innovating networking. Virtualized data centers have e.g. allowed for the global rise of cloud computing and the fast deployment of global web services. As network virtualization allows for resource isolation and thereby enables quality of service guarantees, current research considers how virtualization technology can be applied to ISP and backbone networks to overcome the internet impasse [And+05]. While link virtualization was already allowed for by using IP over MPLS, the idea of Network Functions Virtualization (NFV) currently receives much attention [Eur12]. NFV aims at distributing general computing resources in the network and to deploy services on these so called universal nodes where necessary in a flexible fashion.

Based on the new flexibility offered by network virtualization, this thesis considers how two basic fundamental communivation services can be deployed in virtualized networks, namely multicasting and aggregation. In the well-known multicast communication scheme, a single sender needs to forward (the same) data towards multiple receivers. As implementing multicast via independent communication channels from the sender towards all receivers (unicast) does not scale, existing multicasting technologies rely on local in-network processing on e.g. routers, such that a data stream is duplicated only where necessary. Efficient multicasting is essential for a wide range of applications, e.g. video and audio conferencing, optical multicast for IPTV and synchronization of replicated services.

In contrast to multicasting, the aggregation communication scheme asks for forwarding data of multiple senders towards a single receiver. Assuming that the single receiver shall compute an associative and commutative function, processing nodes that receive multiple data streams can compute intermediate results and only forward the result. This again allows for reducing the bandwidth needed overall as well as spreading the computational load accross multiple nodes. The general aggregation communication scheme is of importance in several different contexts. It can be applied directly to sensor networks where values or even messages can be aggregated to reduce e.g. energy consuption but may also be used in data centers and ISP networks. In data centers, MapReduce is often times used to distribute big data applications across many nodes. Aggregation can here be used during the shuffle phase to locally reduce incoming data. Considering ISP networks, a common task is to perform network analytics to e.g. trace errorneous configurations or adapt load balancing. As the amount of data transferred via the network is non-negligible, aggregation could be used on processing nodes to filter unrelevant information.

To implement multicasting and aggregation efficiently and in a scalable fashion, in-network processing is essential. However, while allowing for drastically reducing the consumed bandwidth, it also comes at a certain cost, which may be either monetary or opportunistic. The concept of NFV enabled ISP networks envisions universal nodes that are shared across multiple services, such that deploying each service on all universal nodes is not feasible. Therefore,

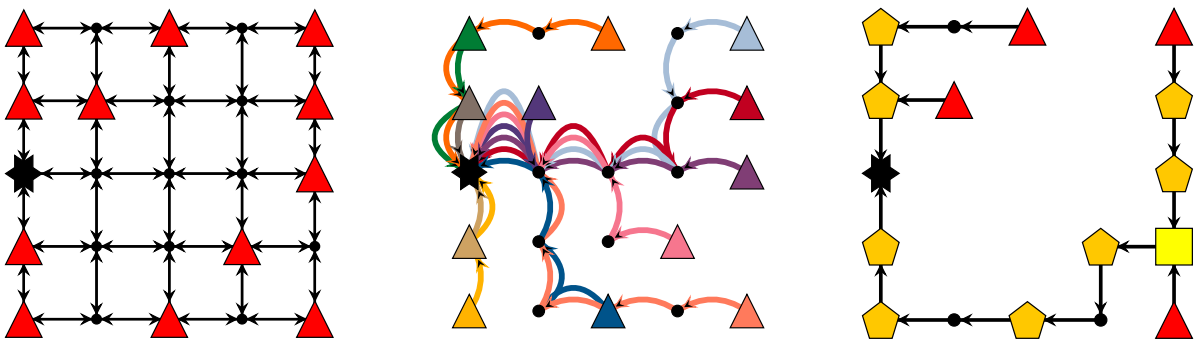
using a certain universal node comes at the opportunistic cost, that other services may not use this node. Another significant type of opportunistic cost is the overall scalability of the deployed service. By using in-network processing functionality on all nodes, bandwidth used for the application can be reduced to a minimum. However, as most applications require stateful communication, using in-network processing on all nodes may impede scalability due to the overhead associated with keeping state. Lastly, real monetary costs arise e.g. when deploying multicast enabled optical routers or universal nodes in ISP networks, and in general for management and service of the hardware.

Besides costs for installing in-network processing functionality, furthermore capacities on processing nodes as well as on links will are considered. The node capacities of processing nodes will effectively upper bound the number of streams a processing node can *process*, such that in the multicasting scenario an incoming stream cannot be duplicated and forwarded to arbitrary many recipients. In the aggregation scheme, the node capacities will limit the number of streams a processing node can *merge*. While the processing nodes' capacacities can be used to model *computational limitations*, the usage of link capacities is necessary in the context of virtualized networks to safeguard resource isolation.

Despite its importance, the underlying optimization problem that asks for trading-off bandwidth and the cost for deploying or using in-network processing functionality while respecting node and link capacities, was not studied before. Indeed, this thesis introduces the first correct and concise graph-theoretic definition of the Constrained Virtual Steiner Arborescence Problem (CVSAP) [1].

In the following the model proposed in this thesis will be discussed in more depth. In Section 1.2 we outline possible applications for our novel model. Lastly, Sections 1.3 and 1.4 give an overview over the thesis and its contribution, respectively

[1] Oliviera and Pardalos consider a similar problem in a series of works, the latest being [OP11]. However, their definition is inherently flawed as it relies on an incorrect Integer Programming Formulation [RS13b].



(a) Aggregation scenario (b) Unicast solution (c) Steiner arborescence

Figure 1.1.: An aggregation example on a 5×5 grid. Senders are depicted as triangles while the receiver is depicted as star. Processing locations are pictured as squares or, in case that a processing location is collocated with a sender, pentagons. In Figure (b), equally colored and dashed edges represent paths, originating at the node with the same color.

1.1. Model Outline

To illustrate our model, consider the aggregation example depicted in Figure 1.1a. Given is a bi-directed 5×5 grids, where triangles represent senders and the star represents the single receiver. The black dots represent further nodes, which are neither receiver nor sender but on which processing functionality may be placed to merge multiple incoming data streams. If no such functionality is placed, then the node must forward all incoming data streams. The task is to connect all senders to the single receiver.

Figure 1.1b presents a simple unicast solution where each sender forwards its information to the receiver directly, such that no processing functionality needs to be installed at all. Such a solution can be computed using known shortest paths algorithms, or minimum cost flow algorithms if capacities are given. Note that the solution uses 41 edges overall.

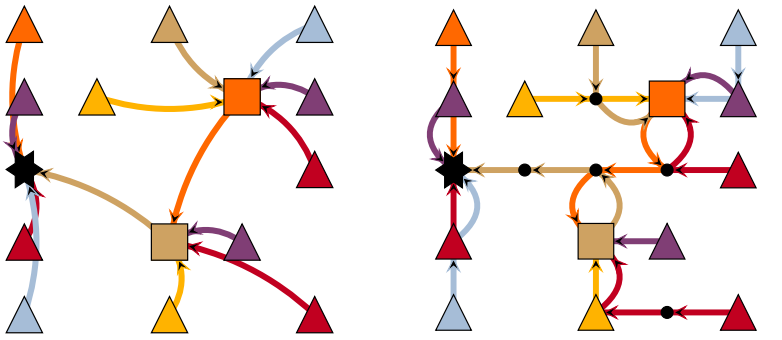
In contrast, when processing functionality can be placed at all nodes and the task is to minimize the bandwidth usage, an optimal solution can be computed using Steiner arborescence algorithms (see Figure 1.1c). Note that this solution uses 16 edges and 9 processing nodes.

Interestingly, the question of how to trade-off bandwidth usage and the cost of installing processing functionality was not studied in the context of combinatorial optimization before. Thus, when searching for optimal solutions to either the multicasting or aggregation problem, one could only activate in-network processing at all nodes or deactivate its usage globally. The main contribution therefore lies in answering the following questions:

1. Which structure do solutions have where in-network processing is only enabled at a subset of nodes?

2. How can such solutions be computed to optimize the overall cost, consisting of costs for bandwidth usage and installing processing nodes?

To outline the structure of solutions we search for, consider the solution presented in Figure 1.2a. In this solution the senders either connect to the receiver directly or to one of the two processing nodes. The upper processing node then forwards its aggregated result towards the lower processing node, which in turn sends its aggregated result towards the receiver. We



(a) Virtual arborescence (b) Paths of virtual arborescence

Figure 1.2.: Virtual arborescence solution using only 26 edges and 2 processing nodes.

denote the solution depicted in Figure 1.2a as virtual arborescence, as edges in this tree correspond to paths in the underlying network (see Figure 1.2b). Note that this solution only uses 2 processing nodes instead of 9 with respect to the Steiner arborescence solution and reduces the number of edges with respect to the unicast solution from 41 to 26. Furthermore, note that while the unicast solution was a directed acyclic graph and the Steiner arborescence solution was an arborescence, the paths of the virtual arborescence solution actually contain *cycles*.

Given the notion of virtual arborescences, capacity constraints on processing nodes and edges can be easily modeled. The capacity of processing nodes will effectively limit the degree of activated processing nodes in the virtual arborescence and the links' capacities are enforced by summing up the number underlying paths using an edge (cf. Figure 1.2b).

In the next section, multicasting and aggregation applications will be discussed, to which such virtual solutions will be of importance to.

1.2. Applications

In this section applications are discussed which are sensitive to the trade-off between bandwidth usage and costs for installing processing functionality and therefore motivate the novel approach presented in this thesis. Even though virtualization technologies like software-defined networking and the concept of network functions virtualization are prominent representatives of how virtualized multicast and aggregation communication services might be implemented upon [CB10], our approach has applications in several very different networking contexts, such as backbone, ISP, sensor and data center networks.

Note that all applications presented henceforth require link capacities to be obeyed as otherwise a distinct service degradation would be the consequence. Therefore, only the importance of incorporating processing nodes' capacities to accurately model the applications will be discussed.

1.2.1. Multicast Communication Scheme

Overlay / Virtual Networks

The multicast communication scheme, whose task it is to distribute a single data item to multiple receiver, is widely used throughout the networking literature as it is a generalization of broadcasting (see e.g. [HHR13] for applications to content-addressable networks). It is therefore hardly surprising, that Shi already proposed in 2001 to consider virtual multicast trees to improve scalability of multicasting in the general context of overlay networks [Shi01]. As overlay networks generally operate at the application layer (with respect to the ISO/OSI layer scheme), the paths that are taken to connect nodes cannot be controlled by the user. However, by utilizing e.g. software-defined networking inside a data-center the principle of overlay networks can be generalized to virtual networks.

In the context of overlay and virtual networks, the consideration of processing nodes' capacities is of upmost importance, as processing nodes are not customly designed to e.g. multiplex an incoming stream on several ports without experiencing a certain delay or overhead.

Optical Multicast

A more specific application lies in the distribution of e.g. IPTV over optical backbones using technologies as Synchronous Digital Hierarchy (SDH) and reconfigurable optical add / drop multiplexer (ROADM) [Her+07]. ROADMs are used to inject or extract information to or from a 'wavelength' such that the information can be distributed to the access network over e.g. ethernet. In the context of our in-network processing terminology, processing nodes would need to represent ROADMs, such that flows can be duplicated and destined to other destinations seamlessly. Finding the best locations for these expensive devices is an important network design problem [Hu+04]. Furthermore, dropping 'packets' at ROADMs and 'splitting' a wavelength induces a distinct power loss of the signal [Rou03]. Therefore, such optical multicast enabled routers are only able to forward the stream towards a limited number of recipients. This necessitates the consideration of capacities for processing nodes.

Geo-Replication of Services / Caching

Another important application stems from the current trend in geo-replication of services and cache placement. In this context the general task is to distribute copies of data throughout the network to balance load across servers and to reduce e.g. the users' latency of accessing for example video streams [Nar+13]. The relation to the multicasting communication scheme was already established by Oliveira and Pardalos [OP11], who defined the Flow Streaming Cache Placement Problem (FSCPP). In the FSCPP processing nodes represent caches connected in an arbitrary hierarchy and receivers are users accessing content. Even though they generally consider a similar problem, their problem definition is flawed [RS13b] and only the restrictive model is considered, which does not consider edge costs.

As in geo-replication scenarios the amount of transferred data is possibly very high and a tight synchronization of the sending nodes with its recipients is necessary, processing node capacities need to be taken into consideration. We furthermore note that by limiting the number of recipients, a faster fail-over might be possible, in case the sender fails.

1.2.2. Aggregation Communication Scheme

The aggregation communication scheme similarly has many different applications ranging over different kinds of networks and employing different (virtualization) technologies. As the problem formulation presented in this thesis allows for arbitrary hierarchies of aggregation, it fully applies only to tasks where a commutative and associative function needs to computed. This is for example the case in sensor networks and applies to some extent also to big data applications.

Nevertheless, by limiting the number of incoming streams that a processing node can aggregate, the presented model might be applicable in other contexts as well.

Sensor Networks

In the context of sensor networks, an important task is to minimize the energy consumption. As most energy is consumed during radio communication, models have been proposed

in which multiple messages are first collected and then sent consecutively to minimize the time the radio has to be enabled [KEW02]. Furthermore, when computing a commutative and associate function on measurements originating at sensor nodes, the aggregation scheme we propose can be directly adopted [DCX03]. The objective to minimize both bandwidth usage and the cost of installing processing nodes applies to both these applications. In the first case processing enabled sensor nodes might need to be equipped with more storage, thereby increasing their hardware cost. Secondly, as sensor nodes are distributed systems which are inherently subject to external interferences, using to many processing nodes might reduce the system's scalability due to the overhead necessary to (re-)structure the virtual aggregation trees upon failures.

By introducing processing node capacities, the limited computational power of sensor nodes can be adequately modeled.

Big Data Applications

Big data applications make frequent use of aggregation and filtering, such that the question arises where to place such functionality. The authors of [Cos+12] propose for example a direct-interconnect network topology called Camdoop for executing MapReduce tasks. Using this network topology, during the shuffle phase intermediate results are locally merged to reduce the consumed bandwidth. The authors specifically take commutative and associative functions into account and argue that this approach can still significantly reduce traffic, even when the reduce function is not commutative and associative.

Therefore, our approach could e.g. be applied when considering data centers that provide MapReduce-as-a-Service [WS13], where MapReduce jobs can be spawned by customers and the provider could try to place aggregation functionality within the network to reduce link load. Considering such scenarios is motivated by the common oversubscription of links by factors of up to 1:5 [BH09]. As aggregating MapReduce outputs at intermediate nodes comes at the price of installing further virtual machines or processes, the trade-off between traffic and the additional usage of computational power needs to be considered.

Network Analytics

Lastly, the aggregation communication scheme might also be applicable to the management of ISP networks. The authors of [Cra+03] present the Gigascope stream database that was deployed throughout the AT&T network in 2003 to allow for stream based network analysis, to e.g. trace errorneous configurations and unexpected events. Using Gigascope, the user can define (sources of) information and relation between them in an SQL-like fashion to continuously query the global network state. Assuming the ISP's support of NFV and a virtualized transport layer, the aggregation scheme could be used to merge data streams and discard information of no interest near the sources of information, reducing the traffic significantly. Again the motivation for not using in-network processing at each possible location, are both scalability of the system and the limited computational resources in the network.

Note that in this scenario, the ability to merge multiple incoming data streams into a single one depends entirely on the semantics of the query. However, by introducing capacities of

processing nodes, the aggregation factor is limited at each node.

1.3. Overview

This thesis is subdivided into five parts. Part I introduces the Constrained Virtual Steiner Abrorescence Problem (CVSAP) together with a set of weaker variants. The computational complexity of CVSAP is studied and its inapproximability is shown. Based upon relationships among weaker variants and well-known optimization problems, approximation algorithms are obtained for three of the five variants.

In Part II two different Integer Programming (IP) formulations to solve CVSAP to optimality are presented. While the first one uses a multi-commodity flow formulation to represent paths explicitly, a single-commodity flow formulation is presented that abstracts from the origin and the destination of flows. Based on a novel decomposition scheme, the correctness of the single-commodity formulation is established, yielding the VirtuCast algorithm to solve CVSAP.

Part III introduces a combinatorial as well as multiple linear heuristics for CVSAP which are employed as primal heuristics within our VirtuCast algorithm. The polynomial runtime of the heuristics is proven, thereby complementing the development of the exact, non-polynomial VirtuCast algorithm.

Part IV presents an extensive computational study. Due to the novelty of CVSAP and its wide range of applications, the performance of all presented algorithms is computationally studied on three distinct topologies. For each topology 75 instances are considered, varying both in size, cost and capacity distributions.

This thesis is concluded by summarizing the obtained results and discussing related and future work in Part V.

1.4. Contribution

This thesis initiates the study of the Constrained Virtual Steiner Arborescence Problem, which models many aggregation and multicasting related applications in virtualized networks. To the author's knowledge, no equally general model has been considered so far, and we give the first concise graph-theoretic definition. By showing the inapproximability of CVSAP (unless $P = NP$) and by obtaining polynomial reductions to known optimization problems for variants of CVSAP, the computational complexity of CVSAP is studied to a great extent, yielding approximation algorithms for three of 5 variants.

The theoretical discussion of the computational complexity is complemented by the introduction of two exact Integer Programming formulations and four types of heuristics for CVSAP. Most notably, we prove the correctness of the single-commodity flow formulation by devising a novel flow decomposition algorithm which resolves circularities in polynomial time. This constructive proof does not only allow us to derive the exact VirtuCast algorithm, but also enables the usage of its compact linear relaxation in Linear Programming (LP) based heuristics.

Our extensive computational study, containing more than 3000 experiments and amounting to more than 100 days of wallclock runtime, establishes the following results underlining our theoretical contributions:

1. The naive multi-commodity flow formulation cannot be used to solve (realistically) sized instances *within reasonable time*: on some instances not even the initial (root) relaxation could be computed within one hour. If dual bounds can be established, these are substantially worse than the ones established by utilizing the VirtuCast algorithm.

2. The VirtuCast algorithm, which relies on the novel single-commodity flow formulation, generates high quality dual bounds within minutes. Based on the compactness of the single-commodity formulation, linear relaxations can be computed quickly, enabling the development of LP-based heuristics.

3. The (linear) heuristics proposed in this thesis are highly effective in finding solutions. A clear trade-off between the different heuristics' runtime and the quality of solutions is established. Therefore, depending on the applications' temporal restrictions to find a solution for a CVSAP instance, an appropriate heuristic can be selected.

4. By coupling the VirtuCast algorithm with the LP-based heuristics, an highly effective solver for CVSAP is obtained, which obtains solutions *for all* considered instances of less than 6% of optimality within one hour and achieves a median objective gap of 1.5% on two of the three considered topologies.

Previously Published Results

Some parts of this thesis have been published as joint work with Stefan Schmid in the proceedings of the 17th International Conference On Principles Of Distributed Systems [RS13c] as well as in an extended technical report on arXiv [RS13b]. The author hereby certifies that parts of the previously published works that are used within this thesis were solely written by the author himself. We mark the captions of sections with a star (★) if most of its content has been already published in one of the two above mentioned publications.

Part I.

Theory of CVSAP

2. The Constrained Virtual Steiner Arborescence Problem ★

The Constrained Virtual Steiner Arborescence Problem (CVSAP) considers multicast and aggregation problems in which processing locations can be *chosen* to reduce traffic. As discussed in Section 1.2, installing or leasing in-network processing capabilities comes at a certain (opportunistic or monetary) cost and a trade-off between installing processing functionality and traffic reductions is searched for. In contrast to the classic Steiner Tree Problems [Voß06], our model distinguishes between nodes that merely forward traffic and nodes that may actively *process flows*, i.e. that a priori processing functionality may only be placed on a subset of all the nodes. This is an important feature to accurately model multicast and aggregation tasks, as clearly processing functionality cannot be placed on nodes that the user does not have administrative control over, or that simply do not meet necessary coniditions as e.g. sufficiently enough remaining computational power.

Independent of whether the multicasting or the aggregation case is considered, the task is to construct a minimal cost spanning arborescence on the set of active processing nodes, sender(s) and receiver(s), such that edges in the virtual arborescence correspond to paths in the original graph. As edges in the virtual arborescence represent logical links (i.e., routes) between nodes, we refer to the problem as *Virtual* Steiner Arborescence Problem. Based on the notion of virtual edges, the underlying paths may overlap and may use both the (resource-constrained) nodes and edges in the original graph multiple times (cf. Figure 1.2).

This section is structured as follows. In Section 2.2 the Constrained Virtual Steiner Arborescence and its variantes will be introduced. In Section 2.3 it is shown that finding a feasible solution to CVSAP is NP-complete and therefore, unless $P = NP$, CVSAP cannot be approximated. Motivated by this result, in Section 2.4 several weaker variants are introduced and first reductions between these derived.

2.1. Notation

In a directed graph $G = (V_G, E_G)$ we denote by $\mathcal{P}_G$ the set of all simple, directed paths in G. Given a set of simple paths $\mathcal{P}$, we denote by $\mathcal{P}[e]$ the subset of paths contained in $\mathcal{P}$ that contain edge e. We use the notation $P = \langle v_1, v_2, \ldots, v_n \rangle$ to denote the directed path P of length $|P| = n$ where $P_i \triangleq v_i \in V_G$ for $1 \leqslant i \leqslant n$ and $(v_i, v_{i+1}) \in E_G$ for $1 \leqslant i < n$. We denote the set of outgoing and incoming edges, restricted on a subset $F \subseteq E_G$, for node $v \in V_G$ by $\delta_F^+(v) = \{(v,u) \in F\}$ and $\delta_F^-(v) = \{(u,v) \in F\}$ and set $\delta_F^+(W) = \{(v,u) \in F | v \in W, u \notin W\}$ and respectively $\delta_F^-(W) = \{(u,v) \in F | v \in W, u \notin W\}$. We abridge $\mathsf{f}((\mathsf{y},\mathsf{z}))$ to $\mathsf{f}(\mathsf{y},\mathsf{z})$ for functions defined on tuples.

2.2. Definition of the Constrained Virtual Steiner Arborescence Problem

Our general problem definition presented henceforth captures both the multicast and the aggregation scenario. As the CVSAP is closely related to the Steiner Arborescence Problem (SAP), we adopt and extend the Steiner literature terminology. We denote the single sender or receiver as *root* and the receivers or senders that need to be connected as terminals. Nodes that can be equipped with processing functionality are called *Steiner sites*. Upon installation of processing functionality, we refer to these nodes as *activated Steiner nodes*.

We model the physical infrastructure as capacitated, directed network $G = (V_G, E_G, c_E, u_E)$ with integral capacities on the edges $u_E : E_G \to \mathbb{N}$ and positive edge costs $c_E : E_G \to \mathbb{R}^+$. On top of this network, we define the following abstract communication request.

Definition 2.1: ABSTRACT COMMUNICATION REQUEST
An abstract communication request on a graph G is defined as a 5-tuple $R_G = (r, S, T, u_r, c_S, u_S)$, where

- $T \subseteq V_G$ is the set of terminals,
- $r \in V_G \backslash T$ denotes the root with integral capacity $u_r \in \mathbb{N}$ and
- $S \subseteq V_G \backslash (\{r\} \cup T)$ denotes the set of possible *Steiner sites* with associated activation costs $c_S : S \to \mathbb{R}^+$ and integral capacities $u_S : S \to \mathbb{N}$.

It must be noted that we require the sets S and T to be disjoint for terminological reasons and that *joint* terminals and Steiner sites can be modeled using Construction 2.7.

In the aggregation scenario the terminals represent nodes holding data that needs to be forwarded to the root (the single receiver) while data may be aggregated at active Steiner nodes. Contrary, in the multicast scenario the root represents the single sender that must stream (the same) data to each of the terminals, while active Steiner nodes may duplicate and reroute the stream. The capacities on the root and on the Steiner sites will limit the degree in the virtual arborescence, formally introduced next.

Definition 2.2: VIRTUAL ARBORESCENCE
Given a directed graph $G = (V_G, E_G)$ and a root $r \in V_G$, a *virtual arborescence (VA)* on G is defined as $\mathcal{T}_G = (V_\mathcal{T}, E_\mathcal{T}, r, \pi)$, where $\{r\} \subseteq V_\mathcal{T} \subseteq V_G$, $E_\mathcal{T} \subseteq V_\mathcal{T} \times V_\mathcal{T}$ and $\pi : E_\mathcal{T} \to \mathcal{P}_G$ maps each edge in the arborescence on a simple directed path $P \in \mathcal{P}_G$ such that

(VA-1) $(V_\mathcal{T}, E_\mathcal{T}, r)$ is an arborescence root at r with edges either directed towards or away from r,

(VA-2) for all $(u, v) \in E_\mathcal{T}$ the directed path $\pi(u, v)$ connects u to v in G.

A link $(v, w) \in E_{\mathcal{T}}$ represents a logical connection between nodes v and w while the function $\pi(v, w) = P$ defines the route taken to establish this link. Note that the directed path P must, *pursuant* to the orientation (v, w) of the logical link in the arborescence, start with v and end at w. Figure 1.2 illustrates our definition of the VA: equally colored and dashed paths represent edges of the Virtual Arborescence. Using the concept of virtual arborescence, we can concisely state the problem we are attending to.

Definition 2.3: CONSTRAINED VIRTUAL STEINER ARBORESCENCE PROBLEM
Given a directed capacitated network $G = (V_G, E_G, c_E, u_E)$ and a request $R_G = (r, S, T, u_r, c_S, u_S)$ as above, the *Constrained Virtual Steiner Arborescence Problem (CVSAP)* asks for a minimal cost Virtual Arborescence $\mathcal{T}_G = (V_{\mathcal{T}}, E_{\mathcal{T}}, r, \pi)$ satisfying the following conditions:

(CVSAP-1) $\{r\} \cup T \subseteq V_{\mathcal{T}}$ and $V_{\mathcal{T}} \subseteq \{r\} \cup S \cup T$,

(CVSAP-2) for all $t \in T$ holds $\delta^+_{E_{\mathcal{T}}}(t) + \delta^-_{E_{\mathcal{T}}}(t) = 1$,

(CVSAP-3) for the root $\delta^+_{E_{\mathcal{T}}}(r) + \delta^-_{E_{\mathcal{T}}}(r) \leqslant u_r$ holds,

(CVSAP-4) for all $s \in S \cap V_{\mathcal{T}}$ holds $\delta^-_{E_{\mathcal{T}}}(s) + \delta^+_{E_{\mathcal{T}}}(s) \leqslant u_S(s) + 1$ and

(CVSAP-5) for all $e \in E_G$ holds $|\,(\pi(E_{\mathcal{T}}))\,[e]| \leqslant u_E(e)$.

Any VA $\mathcal{T}_G$ satisfying CVSAP-1 - CVSAP-5 is said to be a feasible solution. The cost of a Virtual Arborescence is defined to be

$$C_{\text{CVSAP}}(\mathcal{T}_G) = \sum_{e \in E_G} c_E(e) \cdot |\,(\pi(E_{\mathcal{T}}))\,[e]| + \sum_{s \in S \cap V_{\mathcal{T}}} c_S(s)\,,$$

where $|\,(\pi(E_{\mathcal{T}}))\,[e]|$ is the number of times an edge is used in different paths.

In the above definition, CVSAP-1 states that terminals and the root must be included in $V_{\mathcal{T}}$, whereas non Steiner sites are excluded. We identify $V_{\mathcal{T}} \backslash (\{r\} \cup T)$ with the set of active Steiner nodes. Condition CVSAP-2 states that terminals must be leaves in $\mathcal{T}_G$ and CVSAP-3 and CVSAP-4 enforce degree constraints in $\mathcal{T}_G$. The term $\pi(E_{\mathcal{T}})$ in Condition CVSAP-5 determines the set of all used paths and consequently $\pi(E_{\mathcal{T}})[e]$ yields the set of paths that use $e \in E_{\mathcal{T}}$. As π is injective and maps on simple paths, Condition CVSAP-5 enforces that edge capacities are not violated.

Definition 2.4: MULTICAST / AGGREGATION CVSAP
The CVSAP constitutes the *Constrained Virtual Steiner Aggregation Problem (A-CVSAP)* in case that the edges of $E_{\mathcal{T}}$ are oriented towards r, or conversely constitutes the *Constrained Virtual Steiner Multicast Problem (M-CVSAP)* if edges are oriented away from r. We denote the set of feasible solutions by $\mathcal{F}_{\text{A-CVSAP}}$ and $\mathcal{F}_{\text{M-CVSAP}}$ respectively.

It is important to note that Definitions 2.2 and 2.4 together imply that in A-CVSAP each terminal is connected to the root and conversely that in M-CVSAP the root is connected to all terminals. Furthermore, the following remark establishes that the degree constraints (see CVSAP-3 and CVSAP-4) effectively constrain the number of incoming connections (A-CVSAP) and respectively constrain the number of outgoing connections (M-CVSAP).

Remark 2.5 *(Degree Constraints).* Note that in case of M-CVSAP the conditions CVSAP-2 amd CVSAP-3 reduce to $\forall s \in S \cap V_{\mathcal{T}}.\delta^{+}_{E_{\mathcal{T}}}(s) \leqslant u_S(s)$ and $\delta^{+}_{E_{\mathcal{T}}}(r) \leqslant u_r$ respectively. Analogously, conditions CVSAP-2 and CVSAP-3 simplify to $\forall s \in S \cap V_{\mathcal{T}}.\delta^{-}_{E_{\mathcal{T}}}(s) \leqslant u_S(s)$ and $\delta^{-}_{E_{\mathcal{T}}}(r) \leqslant u_r$ in A-CVSAP.

As the single difference between A-CVSAP and M-CVSAP is the orientation of the edges, the problems can be reduced on each other:

Lemma 2.6: *(Equivalence of A-CVSAP and M-CVSAP)* The problems A-CVSAP and M-CVSAP can be reduced on each other.

Proof: We only sketch the construction of reducing M-CVSAP onto A-CVSAP. Given a M-CVSAP instance $R_G = (r, S, T, u_r, c_S, u_S)$ on the network $G = (V_G, E_G, c_E, u_E)$, first construct the reversed graph $G^R = (V_G, E^G_G, c^G_E, u^G_E)$, where $E^R_G = \{(v,u)|(u,v) \in E_G\}$ and $c^R_E(v,u) \triangleq c_E(u,v)$ and $u^R_E(v,u) \triangleq u_E(u,v)$ for all edges $(u,v) \in E_G$.

Solving the A-CVSAP instance R_G on G^R then yields the virtual arborescence $\mathcal{T}^R_G = (V_{\mathcal{T}}, E^R_{\mathcal{T}}, r, \pi^R)$. By reversing all edges $E^R_{\mathcal{T}}$ and the corresponding paths π^R, the virtual arborescence $\mathcal{T}$ is constructed, which in fact is a solution to the original M-CVSAP instance R_G on G. Note that this is a polynomial and even linear reduction. ■

Based on the above lemma, it suffices to give an algorithm for either one of the two problems. Before showing the inapproximability of CVSAP in the next section, we conclude by giving the following construction to model nodes that are both terminals and Steiner sites.

Construction 2.7: MODELING JOINT TERMINALS AND STEINER SITES

For the sake of identifying Steiner sites that are included in the virtual arborescence with activated Steiner nodes, Definition 2.1 does require S and T to be disjoint.

However, a node $v \in S \cap T$ can easily be modeled in the aggregation scenario, by introducing a new node $v_T \in T$ and letting $v \in S$, such that v_T is only connected to v with $c_E(v, v_T) = 0$ and $u_E(v, v_T) = 1$. Similarly, in the multicast scenario, the newly introduced edge would need to be reversed towards v_T. Figure 2.1 depicts the construction for a single node $v \in S \cap T$ in the aggregation case.

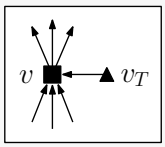


Figure 2.1.

2.3. Inapproximability of CVSAP

The following result shows that deriving polynomial time approximation algorithms for CVSAP is impossible, unless $P = NP$.

Theorem 2.8: ***Inapproximability of CVSAP***

> *Checking whether a feasible solution for CVSAP exists is NP-complete. Thus, unless $NP \subseteq P$ holds, there cannot exist an (approximation) algorithm yielding a feasible solution in polynomial time.*

Proof: We give a reduction from the decision variant of set cover. Let U denote the universe of elements and let $\mathcal{S} \subseteq 2^U$ denote a family of sets covering U. To check whether a set cover using at most k many sets exists, we construct the following CVSAP instance. We introduce a terminal t_u for each element $u \in U$ and a Steiner site s_S for each $S \in \mathcal{S}$. A terminal t_u is connected by a directed link to each Steiner site s_S iff. $u \in S$. Each Steiner site s_S is connected to the root r. We set the capacity of the root to k and capacities of Steiner sites to $|U|$. It is easy to check that there exists a feasible solution to this CVSAP instance iff. there exists a set cover of less than k elements. ■

Notably, the above construction did not rely on node capacities.

Corollary 2.9: *(Inapproximability without node capacities)* In the construction of Theorem 2.8 only the capacity of the root was used, as the capacity of Steiner sites was de facto unbounded. The limited number of outgoing connections of the root can be expressed by introducing a super root, which connects with a single edge to the original root in the above construction. By limiting the number of flow on this edge to be less than u_r, the same inapproximability results holds, when node capacities are removed altogether. ■

The above corollary shows that the inapproximability only relies on the directed nature of the problem, edge capacities and the fact that not all nodes are Steiner nodes.

2.4. Variants of CVSAP

Based on the inapproximability shown in the above section, the consideration of weaker variants of CVSAP and their computational hardness is of interest, too. Based on Corollary 2.9, introducing edge capacities, renders the problem inapproximable. We therefore, introduce the two following (directed) variants of CVSAP, that both do not consider edge capacities.

Definition 2.10: VIRTUAL STEINER ARBORESCENCE PROBLEM
By not considering edge capacities in the underlying network, removing root and Steiner site capacities in the request, and by dropping the node and edge capacity constraints CVSAP-2 - CVSAP-5 the (unconstrainted) Virtual Steiner Arborescence Problem (VSAP) is defined.

Definition 2.11: NODE CONSTRAINED VIRTUAL STEINER ARBORESCENCE PROB. By not considering edge capacities in the underlying network and dropping the edge capacity constraint CVSAP-5 the Node Constrained Virtual Steiner Arborescence Problem (NVSAP) is defined.

Due to its close relation to the Steiner Arborescence Problem, it is natural to consider also the undirected variants of (N/C)VSAP.

Definition 2.12: CONSTRAINED VIRTUAL STEINER TREE PROBLEM
Analogously to the definition of VA, the concept of an undirected (rooted) Virtual Tree can be introduced, in which (undirected) virtual edges are mapped on undirected simple paths (see VA-1), and the orientation constraint VA-2 is dropped. Substituting $\delta^+_{E_\mathcal{T}}(\cdot) + \delta^-_{E_\mathcal{T}}(\cdot)$ by its undirected counterpart $\delta_{E_\mathcal{T}}(\cdot)$ in Definition 2.3, the Constrained Virtual Steiner *Tree* Problem (CVSTP) is defined.

Definition 2.13: (NODE CONSTRAINED) VIRTUAL STEINER TREE PROBLEM
Analog to the definition of the NVSAP and the VSAP (see Definitions 2.11 and 2.10), their undirected equivalents, namely the Node Constrained Virtual Steiner Tree Problem (NVSTP) and Virtual Steiner tree Problem (VSTP) can be derived. Their definition can be obtained from the definition of CVSTP (see Definition 2.12) by analogously dropping edge (and node) capacity constraints.

Due to space constraints, we only note that especially the CVSTP might be of interest in its own right as it might have applications in e.g. designing FTTx access networks (cf. e.g. [BLM13]). However, as Lemma 2.16 proves, CVSTP can be reduced to CVSAP and we therefore all our results apply to CVSTP, too.

Having defined the unconstrained and undirected variants of CVSAP, we establish two simple relations between these variants.

First note the following observation.

Observation 2.14: CVSAP is a strict generalization of NVSAP and VSAP is a strict generalization of VSAP. Analogously, CVSTP is a generalization of NVSTP and NVSTP is a generalization of VSTP.

The second observation, is that the directed variants are generalizations of the undirected variants. To show this property, we use Construction 2.15 to convert an undirected graph into an equivalent directed graph to solve the corresponding directed problem.

The idea exhibited in Construction 2.15 is to replace each undirected edge $e = \{u, v\} \in E_G$ by a gadget of two nodes and five edges, such that all flow sent via e in the undirected network must traverse a single directed edge in G^D. This kind of construction is necessary, if edge capacities are defined on the original network G. If no edge capacities are given, an undirected

Construction 2.15: MODELING UNDIRECTED AS DIRECTED NETWORKS G^D
Given an undirected, network $G = (V_G, E_G, c_E, u_E)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$ and edge capacities $u_E : E \to \mathbb{N}$, we construct the following equivalent directed capacitated network $G^D = (V_G^D, E_G^D, c_E^D, u_E^D)$, with $u_E^D : E_G^D \to \mathbb{N} \cup \{\infty\}$ and $c_E^D : E_G^D \to \mathbb{R}_{\geqslant 0}$, where

(G^D-1) $V_G^D \triangleq V_G \cup \{\underline{uv}^-, \underline{uv}^+ | \{u, v\} \in E_G\}$

(G^D-2) $E_G^D \triangleq \begin{array}{l} \{(\underline{uv}^+, \underline{uv}^-) | \{u, v\} \in E_G\} \\ \cup \{(u, \underline{uv}^+), (v, \underline{uv}^+) | \{u, v\} \in E_G\} \\ \cup \{(\underline{uv}^-, u), (\underline{uv}^-, v) | \{u, v\} \in E_G\} \end{array}$

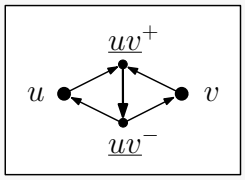


Figure 2.2.

(G^D-3) $u_E^D(\underline{uv}^+, \underline{uv}^-) \triangleq u_E(\{u, v\})$ for $\{u, v\} \in E_G$ and $u_E^D(u, v) = \infty$ else.

(G^D-4) $c_E^D(\underline{uv}^+, \underline{uv}^-) \triangleq c_E(\{u, v\})$ for $\{u, v\} \in E_G$ and $c_E^D(u, v) = 0$ else.

Figure 2.2 depicts the construction for a single undirected edge $\{u, v\}$.

edge can of course be replaced by two directed edges of equal cost. Without a detailed proof, we give the following lemma.

Lemma 2.16: *(CVSTP can be reduced on CVSAP)* Each CVSTP instance, consisting of an undirected network $G = (V_G, E_G, c_E, u_E)$ and a request $R_G = (r, S, T, u_r, c_S, u_S)$, can be reduced on an equivalent CVSAP problem.

Proof: To solve this instance using CVSAP first the directed Network $G^D = (V_G^D, E_G^D, c_E^D, u_E^D)$ is constructed according to Construction 2.15.

Without loss of generality we compute a solution to A-CVSAP on G^D given the request R_G, yielding a (directed) virtual arborescence $\mathcal{T}_{G^D} = (V_\mathcal{T}^D, E_\mathcal{T}^D, r, \pi^D)$. We claim that the (undirected) virtual tree $\mathcal{T}_G = (V_\mathcal{T}, E_\mathcal{T}, r, \pi)$, with $V_\mathcal{T} \triangleq V_\mathcal{T}^D$, $E_\mathcal{T} \triangleq \{\{u, v\} | (u, v) \in E_\mathcal{T}^D\}$ and $\pi(\{u, v\}) \triangleq \pi^D(u, v) \cap V_G$ for all $(u, v) \in E_\mathcal{T}^D$, i.e. that nodes not contained in the original graph are removed from paths, is an optimal solution to the original CVSTP instance.

As $V_\mathcal{T}^D$ is a solution to the corresponding A-CVSAP instance, $V_\mathcal{T}$ fulfills the following properties:

1. $V_\mathcal{T}$ is indeed a virtual tree, as $V_\mathcal{T}^D$ did connect all terminals with the root.
2. $V_\mathcal{T}$ respects the given node capacities, as these hold for $V_\mathcal{T}^D$.
3. $V_\mathcal{T}$ respects the edge capacities. Assume that $V_\mathcal{T}$ violates the capacity of any edge $e = \{u, v\}$ by using it $f(e) > u_E(e)$ many times. As π^D does only map on simple paths, the edge $(\underline{uv}^+, \underline{uv}^-)$ must have been used $f(e)$ many times. This contradicts that $V_\mathcal{T}^D$ is

a feasible solution to the given CVSAP instance as $f(e)$ must be less than or equal to $u_E^D(\underline{uv}^+, \underline{uv}^-) = u_E(\{u, v\})$ by Construction 2.15.

As $V_\mathcal{T}$ has the same cost as $V_\mathcal{T}^D$, $V_\mathcal{T}$ is a feasible solution to the given CVSTP instance of equal cost.

To see that $V_\mathcal{T}$ is indeed optimal, it suffices to check that each (undirected) solution to the given CVSTP instance can be transformed in a similar fashion to a (directed) solution to the corresponding A-CVSAP problem of the same cost. ■

By using the same construction, we can derive the following corollary.

Corollary 2.17: (N)VSTP can be reduced on (N)VSAP. ■

Observation 2.14 and Corollary 2.17 allow us to derive the relations between (N/C)VSAP and (N/C)VSTP depicted in Figure 2.3. In Chapter 3 the relation of VSAP and (N)VSTP to other known optimization problems will be studied, allowing to extend Figure 2.3.

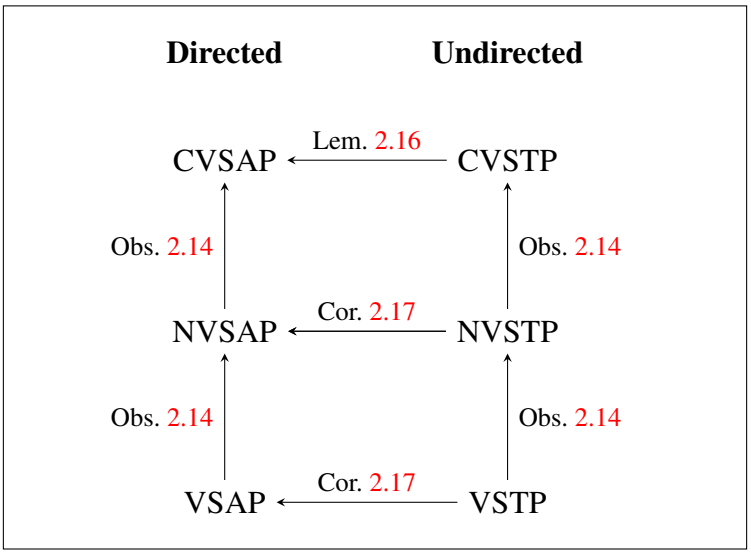


Figure 2.3.: Relation between CVSAP variants. Directed edges represent reductions, such that e.g. there exists a reduction of CVSTP to CVSAP.

3. Approximation of CVSAP Variants

In this section, the weaker (and undirected) variants of CVSAP and their relations to known optimization problems are considered. Based on the found relations, approximation algorithms are derived for VSAP, VSTP and NVSTP. For NVSTP however the approximation algorithm might violate node capacities within a logarithmic factor.

In Section 3.1 a reduction of VSTP onto Connected Facility Location (CFL) is given. Section 3.2 shows the equivalence of VSAP and the Steiner Arborescence Problem (SAP). Lastly, in Section 3.3 a reduction of NVSTP onto the Degree Constrained Node Weighted Steiner Tree Problem (DNSTP) (see Section 3.3. The problems that the CVSAP variants will be reduced to, are introduced in the respective sections. An overview over the relation and reductions shown within this section is given in Figure 3.1.

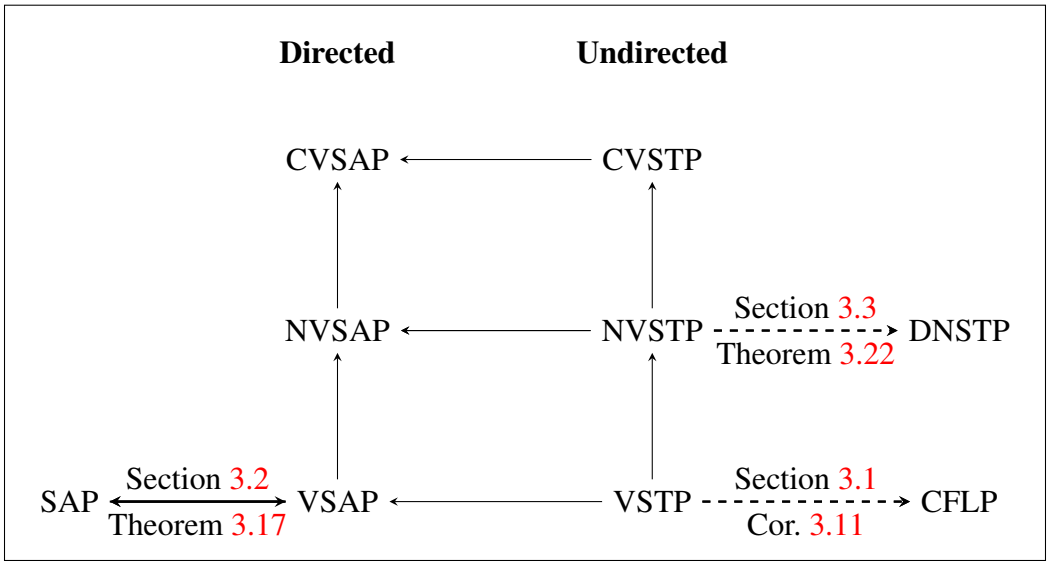


Figure 3.1.: Relation between CVSAP variants and related optimization problems. Solid edges represent strict cost preserving reductions while dashed edges represent reductions preserving the cost upto a constant factor.

3.1. 8-Approximation of VSTP via CFLP

In this section, an approximation algorithm for VSTP is derived based on a reduction onto the Connected Facility Location Problem (CFLP). We first state the common definition of CFLP and then introduce the notion of the shortest path network.

Definition 3.1: CONNECTED FACILITY LOCATION PROBLEM [EIS+10]

Given: Undirected, uncapacitated network $G = (V_G, E_G, c_E)$, facilities $\mathcal{F} \subset V_G$ with associated opening costs $c_{\mathcal{F}} : \mathcal{F} \to \mathbb{R}_{\geqslant 0}$, clients $\mathcal{D} \subset V_G$ with demand $D : \mathcal{D} \to \mathbb{R}_{\geqslant 0}$ and a parameter $M \geqslant 1$.

Task: Determine a subset $F \subseteq \mathcal{F}$ of facilities to open, assign each client $j \in \mathcal{D}$ to some open facility via mapping $\sigma : \mathcal{D} \to F$ and construct a Steiner tree $T_F \subset E_G$ to connect all open facilities F, minimizing the cost

$$C_{\text{CFLP}}(F, T_F, \sigma) = \sum_{i \in F} c_{\mathcal{F}}(i) + M \sum_{e \in T} c_E(e) + \sum_{j \in \mathcal{D}} D(j) \cdot d_G(j, \sigma(j)) \,,$$

where $d_G : V_G \times V_G \to \mathbb{R}_{\geqslant 0}$ is the shortest path distance between nodes in the network G.

3.1.1. Preliminaries

To reduce VSTP onto CFLP we will use the following graph construction, which abstracts from the underlying graph and only represents shortest paths.

Construction 3.2: SHORTEST PATHS NETWORK G_R^{SP}

Given an uncapacitated directed network $G = (V_G, E_G, c_E)$ and a communication request R_G that defines a root $r \in V_G$, the set of Steiner sites $S \subset V_G$ and the set of terminals $T \subset V_G$, we define the shortest paths network $G_R^{SP} = (V_G^{SP}, E_G^{SP}, c_E^{SP})$, with $c_E^{SP} : E_G^{SP} \to \mathbb{R}_{\geqslant 0}$ as follows

(G_R^{SP}-1) $V_G^{SP} \triangleq \{r\} \cup S \cup T$

(G_R^{SP}-2)
$$E_G^{SP} \triangleq \begin{array}{ll} \{(s,r),(t,r) |\ s \in S, t \in T\} & (S \text{ and } T \text{ connect to } r) \\ \cup\, \{(t,s) |\ t \in T,\ s \in S,\ d_G(t,s) < \infty\} & (T \text{ connects to } S) \\ \cup\, \{(s_1,s_2) | s_1, s_2 \in S, d_G(s_1,s_2) < \infty\} & (S \text{ connects to } S) \end{array}$$

(G_R^{SP}-3) $c_E^{SP}(u,v) \triangleq d_G(u,v)$,

where $d_G : V_G \to \mathbb{R}^+ \cup \{\infty\}$ yields the distance, i.e. the length of the shortest paths, between any two nodes of network G or ∞ if no path exists.

Note that the above construction was defined on directed networks, but also extends to undirected networks in the natural way.

Remark 3.3 *(Shortest paths network for undirected networks).*
Given an uncapacitated undirected network $G = (V_G, E_G, c_E)$, the undirected Shortest Paths Network G_R^{SP} is defined as for directed networks, but without edge orientations.

When deriving the approximation algorithm for VSTP, the following observation will be crucial.

Observation 3.4: On undirected graphs, G_R^{SP} contains the complete graph on $\{r\} \cup S$. Furthermore, c_E^{SP} is a metric on G_R^{SP}, such that the triangle inequality

$$c_E^{SP}(\{x,z\}) \leqslant c_E^{SP}(\{x,y\}) + c_E^{SP}(\{y,z\})$$

holds for all $x, y, z \in \{r\} \cup S$ and all connected triangles $\{x,y\}, \{y,z\}, \{x,t\}$ on $\{r\} \cup S \cup T$.

As an important preliminary lemma, we show that solving NVSAP on the shortest paths network G_R^{SP} is equivalent to solving NVSAP on the original graph.

Lemma 3.5: *(Equivalence of NVSAP on G and G_R^{SP})* Given an uncapacitated network $G = (V_G, E_G, c_E)$ and a (directed) NVSAP communication request $R_G = (r, S, T, u_r, c_S, u_S)$, we show that solving NVSAP on G is equivalent to solving it on G_R^{SP}.

Proof: Let $\mathcal{T}_G = (V_\mathcal{T}, E_\mathcal{T}, r, \pi)$ be an optimal solution on G for the given NVSAP instance. We construct a feasible virtual arborescence $\mathcal{T}_{G_R^{SP}} = (V_\mathcal{T}^{SP}, E_\mathcal{T}^{SP}, r, \pi^{SP})$ on G_R^{SP} of equal cost. We set $V_\mathcal{T}^{SP} \triangleq V_\mathcal{T}$ and $V_\mathcal{T}^{SP} \triangleq E_\mathcal{T}$ and $\pi^{SP}(u,v) \triangleq (u,v)$ for all $(u,v) \in E_\mathcal{T}$. As $\mathcal{T}_G$ is a feasible solution connecting all terminals, the same must hold for $\mathcal{T}_{G_R^{SP}}$. Furthermore, the cost of $\mathcal{T}_{G_R^{SP}}$ is equal to the cost of $\mathcal{T}_G$, as $\mathcal{T}_G$ is an optimal solution and therefore $d_G(\pi(u,v)) = d_{G_R^{SP}}(u,v)$ must hold for all $(u,v) \in E_\mathcal{T}$.

Similarly, any optimal solution $\mathcal{T}_{G_R^{SP}}$ on G_R^{SP} can be transformed to a solution on G of the same cost. Hence, solving NVSAP on G_R^{SP} is equivalent to solving it on G.

■

As NVSTP is a generalization of VSTP (see Observation 2.14) and as NVSTP can be reduced onto NVSAP, the above result also holds for VSTP:

Corollary 3.6: Solving VSTP on G is equivalent to solving it on G_R^{SP}. ■

3.1.2. Synopsys of Algorithm ApproxVSTP

Given the above preliminaries, we now present Algorithm ApproxVSTP, which takes as input an undirected network G and an uncapacitated communication request R_G.

After having constructed the shortest paths network an additional node t_0 that is only connected to r is introduced (Line 2). The introduction of this terminal is necessary to guarantee, that r will be contained in the set of opened facilities.

The set of facilities is defined to consist of all Steiner sites S and the root with the corresponding costs (see Line 3) and the set of clients is defined to be the the set of original terminals plus the new node t_0 (see Line 4).

Having computed a solution (F, T_F, σ) to the CFLP instance, a virtual tree on G_R^{SP} is constructed in Lines 6-10. Assuming that $r \in F$ holds, the set of nodes contained are simply all opened facilities and terminals.

The next step is crucial: while the definition of CFLP asks for a Steiner tree connecting F in the underlying network, the definition of VSTP asks for a minimum spanning tree in G_R^{SP} as activated Steiner sites must be connected via independent paths. Thus, in Line 7 such a minimum spanning tree is computed on the subgraph containing only facilities and the set of edges and the corresponding paths are set accordingly in Lines 8 and 9. Lastly, from the virtual tree $\hat{\mathcal{T}}_G^{SP}$ defined on G_R^{SP} a virtual tree on the original network G is obtained by expanding path mappings of the function $\hat{\pi}^{SP}$ to the underlying shortest paths in G.

In the next section it will be proven that ApproxVSTP constructs a feasible solution to VSTP and that the objective value of the found solution is within a factor of 2 with respect to the objective of the CFLP solution. Together with the 4-approximation algorithm for CFLP by Eisenbrand et al. [Eis+10] this will yield an 8-approximation algorithm for VSTP.

Algorithm 3.1: ApproxVSTP

Input : Undirected and uncapacitated network $G = (V_G, E_G, c_E)$,
uncapacitated communication request $R_G = (r, S, T, c_S)$
Output: Feasible Virtual Tree $\hat{\mathcal{T}}_G$ for VSTP

1 **construct** undirected G_R^{SP} (see Construction 3.2)
2 **add** node t_0 to V_G^{SP} **and** edge $\{t_0, r\}$ to E_G^{SP}
3 **set** $\mathcal{F} \triangleq \{r\} \cup S$ **and set** $c_{\mathcal{F}}(s) \triangleq c_S(s)$ for all $s \in S$ **and set** $c_{\mathcal{F}}(r) \triangleq 0$
4 **set** $\mathcal{D} \triangleq \{t_0\} \cup T$ **and** $D(t) \triangleq 1$ for all $t \in \{t_0\} \cup T$
5 **compute** solution (F, T_F, σ) to CFLP instance $(G_R^{SP}, \mathcal{F}, c_{\mathcal{F}}, \mathcal{D}, D, M \triangleq 1)$
6 **set** $\hat{V}_{\mathcal{T}}^{SP} \triangleq F \cup T$
7 **compute** minimum spanning tree $\mathcal{M} \subset E_G^{SP}[F]$ connecting F
8 **set** $\hat{E}_{\mathcal{T}}^{SP} \triangleq \mathcal{M} \cup \{\{t, f\} | t \in T, f = \sigma(t)\}$
9 **set** $\hat{\pi}^{SP}(\{u, v\}) \triangleq \{u, v\}$ for all $\{u, v\} \in \hat{E}_{\mathcal{T}}^{SP}$
10 **set** $\hat{\mathcal{T}}_{G_R^{SP}} \triangleq (\hat{V}_{\mathcal{T}}^{SP}, \hat{E}_{\mathcal{T}}^{SP}, r, \hat{\pi}^{SP})$
11 **obtain** virtual tree $\hat{\mathcal{T}}_G \triangleq (\hat{V}_{\mathcal{T}}, \hat{E}_{\mathcal{T}}, r, \hat{\pi})$ on G **from** $\hat{\mathcal{T}}_{G_R^{SP}}$ (by Lemma 3.5)
12 **return** $\hat{\mathcal{T}}_G$

3.1.3. Proof of 8-Approximation

In the following we will prove that by using an approximate solution for CFLP in Line 5 we can obtain an 8-approximation for VSTP. We begin by showing that Algorithm ApproxVSTP constructs a feasible solution

Lemma 3.7: *(The root is contained in F)*
We show that without loss of generality, we can assume that the root r is included in F.

Proof: Assume that r is not included in F. Then t_0 is connected to some $s = \sigma(t_0)$. As t_0 is only directly connected to r with an edge cost of 0, and therefore any shortest path from t_0 to s must traverse r, we may adapt the solution as follows. $F' \triangleq F \cup \{r\}$, $T' \triangleq T \cup \{r, s\}$, $\sigma'(t) \triangleq \sigma(t)$ for all $t \in T$ and $\sigma'(t_0) \triangleq r$. The solution (F', T', σ') has the same objective value, since the cost for connecting t_0 to s is the same as connecting r to s, as we have set $M = 1$. Therefore, we can assume that $r \in F$ holds. ■

Lemma 3.8: *(Algorithm ApproxVSTP constructs a feasible virtual tree $\hat{\mathcal{T}}_G$)*
We show that $\hat{\mathcal{T}}_G^{SP}$ is indeed a feasible virtual tree on G_R^{SP} such that $\hat{\mathcal{T}}_G$ will be feasible virtual tree on G by Lemma 3.5.

Proof: For checking that $\hat{\mathcal{T}}_G^{SP}$ is a feasible solution to VSTP, Conditions CVSAP-1 and CVSAP-2 as well as Condition VA-1 need to be checked. Condition CVSAP-1 holds by Lemma 3.7 as non-Steiner sites are not contained witin V_G^{SP}. As each terminal (client) is connected to exactly one Steiner node (facility), and a minimum spanning tree is computed on F, Conditions CVSAP-2 and VA-1 hold. ■

Lemma 3.9: *(CFLP on G_R^{SP} gives a lower bound for VSTP on G_R^{SP})*
We show that for any optimal solution $(\hat{F}, \hat{T}_F, \hat{\sigma})$ to CFLP on G_R^{SP} and any feasible solution $\hat{\mathcal{T}}_G^{SP}$ on G_R^{SP}

$$C_{\text{CFLP}}(\hat{F}, \hat{T}_F, \hat{\sigma}) \leqslant C_{\text{VSTP}}(\hat{\mathcal{T}}_G^{SP})$$

holds.

Proof: This result is immediate as the only difference between CFLP on G_R^{SP} and VSTP on G_R^{SP} is that opened facilities can be connected by a Steiner tree instead of a minimum spanning tree. As the set of trees to connect facilities (namely Steiner trees) is a superset of the set of trees to connect Steiner sites (namely minimum spanning trees), and as opening and activation costs are equal, CFLP yields a lower bound for VSTP. ■

The following Theorem 3.10 is a corollary of a well-known result which was first established in 1968 by Gilbert and Pollak [KV12, Theorem 20.6].

Theorem 3.10

For Algorithm ApproxVSTP holds $C_{\text{VSTP}}(\hat{\mathcal{T}}_G) \leqslant 2C_{\text{CFLP}}(F, T_F, \sigma)$.

Proof: First note that connection costs of clients D to facilities F equal the costs of connecting terminals T to $F = \{r\} \cup S$ and that both (F, T_F, σ) and $\hat{\mathcal{T}}_G^{SP}$ induce the same facility opening and Steiner activation costs. The only cost difference therefore lies in the connectivity costs of $F = \{r\} \cup S$.

As c_E^{SP} defines a metric on the set of Steiner nodes of G_R^{SP} (see Observation 3.4), the result that was first published by Gilbert and Pollak [KV12, Theorem 20.6] can be applied, showing that the cost of $\mathcal{M}$ is upper bounded by two times the cost of T_F. The underlying idea is

simple. By duplicating each edge in T_F, a eulerian graph H is obtained. By constructing a walk W from H that only contains nodes in F and removing an edge, a spanning tree is constructed with cost less than two times the original Steiner tree T_F. As in Line 7 a *minimum* spanning tree is computed, the connection costs of $\mathcal{M}$ are bounded by two times the cost of T_F, hence proving the theorem. ■

The following corollary yields our 8-approximation of VSTP.

Corollary 3.11: *(8-Approximation of VSTP via Algorithm ApproxVSTP)*
First note that Algorithm ApproxVSTP is indeed a polynomial algorithm. CFLP can be approximated to within a factor of 4 of optimality by the algorithm given by Eisenbrand et al. [Eis+10]. Using this (polynomial) algorithm to construct an approximate solution to CFLP in Line 5 in Algorithm ApproxVSTP, we obtain an 8-approximation algorithm for VSTP by applying Lemma 3.9 and Theorem 3.10. ■

3.2. Equivalence of VSAP and SAP

Having found an 8-approximation for VSTP in the above section, its directed variant, namely VSAP (see Definition 2.10), will now be considered. It is shown that VSAP is equivalent to the (non-virtual) Steiner Arborescence Problem (SAP), which is also referred to as the Directed Steiner Tree Problem (DSTP) in the literature. Based on this equivalence, VSAP can be approximated to within a logarithmic factor. However, as SAP cannot be approximated to within a factor that is strictly less than logarithmic (unless $P = NP$), this result will also pertain to VSAP (and by Corollary 2.17 to NVSAP, too).

We first state the definition of SAP:

Definition 3.12: STEINER ARBORESCENCE PROBLEM [CHA+98]

Given: Directed, uncapacitated network $G = (V_G, E_G, c_E)$, with a set of terminals $T \subset V_G$ and a root $r \in V_G$.

Task: Find a (Steiner) arborescence $A \subseteq E_G$ connecting all terminals to the root, minimizing the cost $C_{\text{SAP}}(A) = \sum_{e \in A} c_E(e)$.

Note that in the literature the SAP is commonly defined in such a way that the root needs to reach all terminals [Cha+98]. Based on Lemma 2.6, which shows the equivalence of finding arborescences directed towards and away from the root, all results shown in the literature carry over to the case when arborescences are directed towards the root.

3.2.1. Preliminaries

As the definition of SAP does not contain any notion of node costs, we first give a commonly used construction to incorporate node costs as edge costs (in directed networks).

Construction 3.13: MODELING NODE AS EDGE COSTS, NETWORK $G^{\pm}$
Given a directed network $G = (V_G, E_G, c_E, c_V)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$ and node costs $c_V : V_G \to \mathbb{R}_{\geqslant 0}$, node costs can be represented as edge costs in the following way. We construct the directed network $G^{\pm} = (V_G^{\pm}, E_G^{\pm}, c_E^{\pm})$, with $c_E^{\pm} : E_G^{\pm} \to \mathbb{R}_{\geqslant 0}$, where

($G^{\pm}$-1) $V_G^{\pm} \triangleq \{v^+, v^- | v \in V_G\}$

($G^{\pm}$-2) $E_G^{\pm} \triangleq \begin{array}{l} \{(u^-, v^+) | (u, v) \in E_G\} \\ \cup \{(v^+, v^-) | v \in V_G\} \end{array}$

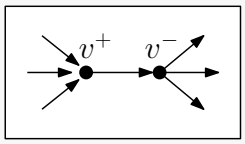


Figure 3.3.

($G^{\pm}$-3) $c_E^{\pm}(u^+, v^-) \triangleq c_E(u, v)$ for $(u, v) \in E_G$ and $c_E^{\pm}(v^+, v^-) = c_V(v)$ for all $v \in V_G$.

Figure 3.3 illustrates the construction for a single node $v \in V_G$.

Using the above construction, the cost of using a node $v \in V_G$ is accounted for, if and only if, the newly introduced edge (v^-, v^+) is used in the solution. As in the case of VSAP edge capacities are not considered, we make again use of the shortest path network (see Construction 3.2). Note that in the directed shortest paths network, terminals have no incoming edges and the root has no outgoing edges. Therefore, applying Construction 3.13 to the shortest paths network will be sufficient to solve VSAP using SAP. We define the combination of both constructions below.

Construction 3.14: NETWORK $G_R^{SP,\pm}$
Given a directed network $G = (V_G, E_G, c_E)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$ and an uncapacitated communication request $R_G = (r, S, T, c_S)$ the network

$$G_R^{SP,\pm} = (V_G^{SP,\pm}, E_G^{SP,\pm}, c_E^{SP,\pm})$$

is obtained by first constructing G_R^{SP} via r, S, T and c_E and then expanding all nodes of G_R^{SP} via Construction 3.13 and the cost function $c_V : V_G^{SP} \to \mathbb{R}_{\geqslant 0}$, $c_V(s) \triangleq c_S(s)$ for all $s \in S$ and $c_V(v) \triangleq 0$ otherwise.

We define $V_G^{SP,+} \triangleq \{v^+ | v \in V_G^{SP}\}$ and $V_G^{SP,-} \triangleq \{v^- | v \in V_G^{SP}\}$.

3.2.2. Proof of Equivalence

In this section we first show how any VSAP instance can be solved using VSAP and then consider the reverse direction.

Lemma 3.15: VSAP can be reduced to SAP.

Proof: Given a directed network $G = (V_G, E_G, c_E)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$ and a VSAP communication request $R_G = (r, S, T, c_S)$, we construct the network $G_R^{SP,\pm}$ (see

Construction 3.14) and compute a Steiner arborescence $A \subset E_G^{SP,\pm}$ (using SAP) to connect the set of terminals $T^- \triangleq \{t^- \in V_G^{SP,\pm} | t \in T\}$ with $r^- \in V_G^{SP,\pm}$.

Given the SAP solution we construct a virtual arborescence $\mathcal{T}_{G_R^{SP}} \triangleq (V_\mathcal{T}^{SP}, E_\mathcal{T}^{SP}, r, \pi^{SP})$ on G_R^{SP} as follows:

- $V_\mathcal{T}^{SP} \triangleq \{v \in V_G^{SP} | v^- \in A\}$, i.e. v is included if and only if node v^- was used in A.
- $E_\mathcal{T}^{SP} \triangleq \{(u,v) \in E_G^{SP} | (u,v) \in A \cap (V_G^{SP,-} \times V_G^{SP,+})\}$.
- $\pi^{SP}(u,v) \triangleq (u,v)$ for all $(u,v) \in E_\mathcal{T}^{SP}$.

It is easy to check, that $\mathcal{T}_{G_R^{SP}}$ is a feasible virtual arborescence on G_R^{SP}, as A itself is a arborescence. We continue by showing that $C_{\text{VSAP}}(\mathcal{T}_{G_R^{SP}}) \leqslant C_{\text{SAP}}(A)$ holds.

For each edge $(u,v) \in E_G^{SP}$ that was included in $E_\mathcal{T}^{SP}$ the SAP solution A accounted for the same cost, namely $c_E^{SP}(u,v)$. Furthermore, only Steiner nodes $s \in S$ have been included in $V_\mathcal{T}^{SP}$, whose outgoing node $s^- \in V_G^{SP,-}$ were used. As s^- can only be reached via s^+ and for using the edge (s^+, s^-) costs of $c_S(s)$ were paid for by the SAP solution, indeed $C_{\text{VSAP}}(\mathcal{T}_{G_R^{SP}}) = C_{\text{SAP}}(A)$ holds. Note that the given construction is polynomial.

■

Lemma 3.16: SAP can be reduced to VSAP

Proof: Showing that SAP can be reduced onto VSAP is simple. Given is an SAP instance on a directed network $G = (V_G, E_G, c_E)$ with the task to connect some set of terminals $T \subset V_G$ to $r \in V_G$. We construct the following VSAP communication request $R_G = (r, S, T, c_S)$, $S \triangleq V_G \backslash \{r\}$ and $c_S(s) \triangleq 0$ for all $s \in S$. As we formally require S and T to be disjoint, we apply Construction 2.7, such that for each node $v \in S \cap T$ a new node v_T is introduced, that directly connects to the original node v. By removing all such nodes $v \in S \cap T$ and placing v_T into T a feasible communication request is derived.

Let $\mathcal{T}_G = (V_\mathcal{T}, E_\mathcal{T}, r, \pi)$ be an optimal solution to the VSAP instance defined in this way. Due to zero cost edges, $\mathcal{T}_G$ may use edges multiple times and even in reverse direction. By the following set of reductions, we can guarantee, that each edge is used at most once and that each node uses at most one ougoing edge.

- If a (directed) edge $(u,v) \in E_G$ is used in multiple paths $\mathcal{P} = \pi(E_\mathcal{T})[(u,v)]$, then all these paths (and the corresponding edges in $E_\mathcal{T}$) can be truncated to end at u. If node u was already an active Steiner node, then the virtual arborescence is still connected, as paths are simple. Otherwise, if node u was not an active Steiner node, it can be activated at zero cost and be placed in $V_\mathcal{T}$. To connect u we pick an arbitrary path $P \in \mathcal{P}$ and proceed as follows. Assuming that node u is the i-th node of P, we connect u towards $P_{|P|}$ via the path $\langle P_i = u, P_{i+1}, \dots, P_{|P|} \rangle$. By definition, the target of the path must either be an active Steiner node s or the root r and in the former case s is still connected to r, since the only possibility to disconnect it would be to create a loop. This however is not possible, as u was not active before and u therefore had no incoming connections.

- If there exists a node $u \in V_G$ for which multiple outgoing edges are used in paths, then the same argument as above can be applied. Let $\mathcal{P} = \pi(\delta^+(u))$ denote the set of paths that use one of u's outgoing edges. If u was an active Steiner node, all paths, except u's own can be truncated to terminate at u and $E_\mathcal{T}$ can be adapted wihtout violating connectivity. On the other hand, if u was not active, then u can be activated and connected in the following way: choose a single path $P \in \mathcal{P}$ and assume again that u occurs at the i-th position of P. Then P_1 can be connected to u via the path prefix $\langle P_1, \dots, P_i \rangle$ and u can be connected to $P_{|P|}$ via the path prefix $\langle P_i, \dots, P_{|P|} \rangle$. Then again, as u is now activated, all other paths using another outgoing edge than u's path can be truncated to end at u.

By iteratively applying the above reduction steps until no further reduction is possible, we can guarantee, that no edge of the underlying graph is used more than once and that for each node at most one outgoing edge is used. As the above reductions preserve connectivity requirements of virtual arborescences, the edge set $A \triangleq \bigcup_{(u,v)\in E_\mathcal{T}} \pi(u,v)$ must be an arborescence in the underlying graph that connects all terminals T to the root, once we remove edges originating at terminals that were artificially introduced by Construction 2.7. As during the reduction process, paths were only truncated or splitted and as activating Steiner nodes comes at no cost, $C_{\text{SAP}}(A) \leqslant C_{\text{VSAP}}(\mathcal{T}_{G_R^{SP}})$ must hold.

To check that the above reduction is indeed polynomial, note that in the optimal solution $\mathcal{T}_G$ each edge of the underlying network G is used at most once by each virtual connection in $E_\mathcal{T}$ as π maps on simple paths. Executing either one of the two local reductions does either strictly decrease the number of edges that are used multiple times or does strictly decrease the number of nodes with more than one outgoing connection while not increasing the other. As finding an edge or a node that is used multiple times or that has multiple outgoing connections can be implemented in polynomial time, the reduction overall is polynomial.

■

By the above two lemmas, we have shown the equivalence of VSAP and SAP.

Theorem 3.17: ***Equivalence of VSAP and SAP.***

There exists a cost preserving mapping between solutions of VSAP and SAP. Therefore, both problems are equivalent.

By the above theorem, all results for SAP pertain to VSAP. The most two important to note are the following ones:

Corollary 3.18: *(Approximation-hardness of VSAP [Cha+98])* As VSAP is equivalent to SAP, there cannot exist an $o(\log |T|)$-approximation algorithm unless $NP \subseteq DTIME[n^{\mathcal{O}(\log \log n)}]$ holds. ■

Corollary 3.19: *($\mathcal{O}(\log |T|)$-approximation for VSAP)* There exists an $\mathcal{O}(\log |T|)$-approximation algorithm for SAP [Cha+98]. Based on the cost-preserving construction given in Lemma 3.15, this approximation result pertains to VSAP. ■

3.3. Approximation of NVSTP via DNSTP

In this section we will lastly derive an approximation algorithm for the Node Constrained Virtual Steiner Tree Problem as introduced by Definition 2.13. Ravi et al. [Rav+01] have considered a very similar problem, namely the Degree-Constrained Node Weighted Steiner Tree Problem (DNSTP), whose definition we repeat below.

Definition 3.20: DEGREE-CONSTRAINED NODE WEIGHTED STEINER TREE PROBLEM [RAV+01]

Given: Undirected network $G = (V_G, E_G, c_E, c_V, u_V)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$[1], node costs $c_V : V_G \to \mathbb{R}_{\geqslant 0}$, and a degree bound function $u_V : V_G \to \mathbb{N}_{\geqslant 2}$ and set of terminals $T \subset V_G$.

Task: Find a Steiner tree $\mathcal{T} \subseteq E_G$ connecting all terminals T, such that for each node v that is contained in $\mathcal{T}$ the degree bound is not violated, i.e. that $\delta_{\mathcal{T}}(v) \leqslant u_V(v)$ holds, minimizing the cost $C_{\text{DNSTP}}(\mathcal{T}) = \sum_{e \in \mathcal{T}} c_E(e) + \sum_{v \in \mathcal{T}} c_V(v)$.

The authors of [Rav+01] have obtained the following important bi-criteria approximation result for DNSTP.

Theorem 3.21: ***Logarithmic bi-criteria approximation for DNSTP [Rav+01]***

There is a polynomial-time algorithm that, given an undirected graph G on n nodes with nonnegative costs on its [edges and][1] nodes, a subset T of nodes called terminals, and a degree bound $u_V(v) \geqslant 2$ for every node v, constructs a Steiner tree spanning all the terminals, with degree $\mathcal{O}(u_V(v) \log |T|)$ at a node v and of cost $\mathcal{O}(\log |T|)$ times that of the minimum-cost Steiner tree of G that spans all the terminals and obeys all the degree bounds.

Based on this approximation result, we will prove the following.

Theorem 3.22: ***Logarithmic bi-criteria approximation for NVSTP***

There is a polynomial-time algorithm that, given a NVSTP instance, constructs a virtual arborescence, for which the degree of included Steiner sites and the root is violated only by a logarithmic factor, and which is of cost $\mathcal{O}(\log |T|)$ times that of the minimum-cost virtual arborescence that satisfies all NVSTP degree constraints.

Note that in the above theorem we state that only the degree constraints of Steiner nodes and the root may be violated. Thus, we will have to enforce that the degree of terminals in the virtual arborescence is one, i.e. that a terminal possesses no processing functionality and is connected to either an active Steiner node or to the root directly.

[1] The original definition and the corresponding theorem only considers the node weighted case. Based on Construction 3.23 which was given by the authors of [Rav+01], edge costs can be included without loss of generality.

3.3.1. Preliminaries

For the sake of completeness, we explicitly state how edge costs can be incorporated into node weighted networks. This construction was already mentioned in [Rav+01].

Construction 3.23: MODELING EDGE COSTS AS NODE COSTS, NETWORK G^N, [RAV+01]
Given an undirected network $G = (V_G, E_G, c_E, c_V)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$ and node costs $c_V : V_G \to \mathbb{R}^+$, edge costs can be represented as node costs in the following way. We construct the undirected network $G^N = (V_G^N, E_G^N, c_V^N)$, with $c_V^N : V_G^N \to \mathbb{R}_{\geqslant 0}$, where

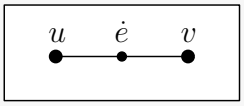


Figure 3.4.

(G^N-1) $V_G^N \triangleq V_G \cup \{\dot{e} | e \in E_G\}$

(G^N-2) $E_G^N \triangleq \{\{u, \dot{e}\}, \{\dot{e}, v\} | e = \{u, v\} \in E_G\}$

(G^N-3) $c_V^N(v) \triangleq c_V(v)$ for $v \in V_G$ and $c_E^{\pm}(\dot{e}) = c_E(e)$ for all $e \in E_G$.

Figure 3.4 illustrates the construction for a single edge $e = \{u, v\} \in E_G$.

As solving a given NVSAP instance given by communication request R_G on G is equivalent to solving it on G_R^{SP} (see Lemma 3.5), and as NVSTP can be reduced to NVSAP by Corollary 2.17, the same result holds in the undirected case as well.

Corollary 3.24: *(Equivalence of NVSTP on G and G_R^{SP})*
Given an uncapacitated, undirected network $G = (V_G, E_G, c_E)$ and a NVSTP communication request $R_G = (r, S, T, u_r, c_S, u_S)$, solving NVSTP on G is equivalent to solving it on G_R^{SP}. ■

The next observation shows how NVSTP can generally be reduced onto DNSTP.

Observation 3.25: *(Reducing NVSTP onto DNSTP)*
Given an undirected, uncapacitated network $G = (V_G, E_G, c_E)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$ and a NVSTP communication request $R_G = (r, S, T, u_S)$, first the shortest paths network G_R^{SP} is constructed.

We define the degree bound function $u_V^{SP} : V_G^{SP} \to \mathbb{N}_{\geqslant 1}$ for DNSTP on G_R^{SP} by $u_V^{SP}(s) \triangleq u_S(s)$ for all $s \in S$, $u_V^{SP}(r) \triangleq u_r(r)$ and $u_V^{SP}(t) \triangleq 1$ for all $t \in T$ and the node cost $c_V^{SP} : V_G^{SP} \to \mathbb{R}_{\geqslant 0}$ as $c_V^{SP}(s) \triangleq c_S(s)$ for all $s \in S$ and $c_V^{SP}(v) \triangleq 0$ for all $v \in V_G^{SP} \backslash S$.

It is easy to check that each solution of DNSTP on the network $(V_G^{SP}, E_G^{SP}, c_E^{SP}, c_V^{SP}, u_V^{SP})$ and the set of terminals $\{r\} \cup T$ represents a feasible solution for the NVSTP instance on G_R^{SP}, as DNSTP naturally defines a virtual tree.

1. All terminals and the root are connected by a Steiner tree. Therefore, the connectivity requirement of Definition VA-1 holds.

2. All degree constraints of NVSTP are enforced by the definition of u_V^{SP}.

3. For each Steiner site $s \in S$ that is included in the DNSTP solution, the cost of $c_S(s)$ is accounted for.

By Corollary 3.24 this feasible solution then yields a feasible solution for the original NVSTP instance on G. Furthermore, both the DNSTP solution and the NVSTP solution will have the same objective value by construction.

While the Observation 3.25 suggests a direct reduction of NVSTP onto DNSTP, such that Theorem 3.21 can be applied to obtain our Theorem 3.22, the following needs to be taken into consideration.

1. By setting $u_V^{SP}(t) = 1$ for terminals $t \in T$, we violate the requirement that $u_V^{SP}(v) \geqslant 2$ holds for all nodes $v \in V_G^{SP}$ (see Definition 3.20 and Theorem 3.21).

2. Furthermore, the solution of the approximation algorithm of Ravi et al. (see Theorem 3.21) may violate degree bounds within a logarithmic factor.

To obtain Theorem 3.22, we therefore have to show how a DNSTP solution in which terminals are connected to multiple nodes can be transformed into a solution of (approximately) the same cost in which terminals are leaves. This algorithm will be introduced in the next section. As main preparation to derive Algorithm Leafify, we will argue about the structure of the connections of terminals using the below definition.

Definition 3.26: TERMINAL CONNECTION SUBGRAPH, $G_{\mathcal{T}}^{\vec{TS}}$
Given a DNSTP solution $\mathcal{T} \subseteq E_G^{SP}$, we define the following subgraph $G_{\mathcal{T}}^{\vec{TS}} = (V_{\mathcal{T}}^{\vec{TS}}, E_{\mathcal{T}}^{\vec{TS}})$ of G_R^{SP}.

($G_{\mathcal{T}}^{\vec{TS}}$-1) $V_{\mathcal{T}}^{\vec{TS}} \triangleq V_G^{SP} \backslash (\{t \in T | \delta_{\mathcal{T}}(t) = 1\} \cup \{s \in S | \delta_{\mathcal{T}}(s) = 0\})$

($G_{\mathcal{T}}^{\vec{TS}}$-2) $E_{\mathcal{T}}^{\vec{TS}} \triangleq \{\{t, v\} \in \mathcal{T} | t \in V_{\mathcal{T}}^{\vec{TS}}\}$

In the above definition, all terminals that already have a degree of 1 are exlcuded and so are unused Steiner sites. Only edges incident to terminals ,with degree greater one, are included.
The following observation follows from the construction of G_R^{SP}.

Observation 3.27: $G_{\mathcal{T}}^{\vec{TS}}$ is a bipartite graph with disjoint node sets $V_{\mathcal{T}}^{\vec{TS}} \cap T$ and $V_{\mathcal{T}}^{\vec{TS}} \backslash T$.

The next observation follows from the fact that $\mathcal{T}$ is a tree.

Observation 3.28: $G_{\mathcal{T}}^{\vec{TS}}$ is a forest and thereby acyclic.

The next lemma will be the key for deriving the desired reduction.

Lemma 3.29: There exists a matching of size $|V_{\mathcal{T}}^{\vec{TS}}|$ in $G_{\mathcal{T}}^{\vec{TS}}$.

Proof: Let $\mathcal{M} \subset E_{\mathcal{T}}^{\vec{TS}}$ denote a matching of maximal cardinality in $G_{\mathcal{T}}^{\vec{TS}}$. As for $|\mathcal{M}| = |V_{\mathcal{T}}^{\vec{TS}}|$ the lemma would hold, assume for the sake of deriving a contradiction, that $|\mathcal{M}| < |V_{\mathcal{T}}^{\vec{TS}}|$ holds, such that there exists a terminal $t_0 \in V_{\mathcal{T}}^{\vec{TS}}$ that is not connected in $\mathcal{M}$.

We use the following iterative construction. We define the following two sets $T_0 = \{t_0\}$ and $S_0 \triangleq \{s | \{t_0, s\} \in E_{\mathcal{T}}^{\vec{TS}}\}$ and iterate according to the following scheme for $i \geqslant 1$.

$$T_i \triangleq \{t | s \in S_{i-1}, \{t, s\} \in \mathcal{M}\}$$
$$S_i \triangleq \{s | t \in T_i, \{t, s\} \in E_{\mathcal{T}}^{\vec{TS}} \backslash \mathcal{M}\}$$

We first note that by definition of S_i and T_i all nodes contained in S_i and T_i are connected to t_0. Furthermore, as $G_{\mathcal{T}}^{\vec{TS}}$ is a forest, the path, with which each node of S_i and T_i is connected to t_0 is unique. As we assume that $\mathcal{M}$ is a maximal matching, the set of terminals $\bigcup_{i \geqslant 0} T_i$ cannot be adjacent to a single Steiner node which is not covered by $\mathcal{M}$: if a terminal $t_i \in T_i$ would exists, such that $\{t_i, s\} \in E_{\mathcal{T}}^{\vec{TS}}$ and $s \notin \mathcal{M}$ holds, then the unique path that connects s via t_i and the sets $S_{i-1}, T_{i-1}, S_{i-2}, \ldots, T_1$ to t_0 could be employed as *augmenting path* to increase the number of edges by one. By the maximality of $\mathcal{M}$, this is not possible. Therefore, each Steiner node $s_i \in S_i$ must be connected to an unique terminal in T_{i+1}. Thus $|T_{i+1}| = |S_i|$ holds. However, if s_i were to be covered by node $t_j \in T_j$ for $j < i - 1$ then a cycle consisting of nodes in $(T_j, S_j, T_{j+1}, S_{j+1}, \ldots, S_i, T_i)$ would have existed in $G_{\mathcal{T}}^{\vec{TS}}$ before as all nodes are uniquely connected to t_0 and s_i were to be reachable via two paths. Lastly, as each terminal has degree at least two and can only be covered by at most one edge of $\mathcal{M}$, $|S_i| \geqslant |T_i|$ follows as no two Steiner nodes within T_i can be connected to the same node $s \in S_i$, since this would again induce a cycle. By the above observations, therefore

$$|T_0| \leqslant |S_0| \leqslant |T_1| \leqslant |S_1| \leqslant \ldots$$

must hold. However, as the sets S_i, S_j are disjoint for $i \neq j$ and as $|T_0| = 1$ holds, this implies that $|S_i| \geqslant i$ holds for all $i \geqslant 0$. As we are considering finite graphs and the set of nodes is finite, this cannot be possible true. Therefore, there always exists a matching of size $|V_{\mathcal{T}}^{\vec{TS}}|$ in $G_{\mathcal{T}}^{\vec{TS}}$. ■

Based on the above lemma, it is easy to construct Algorithm Leafify.

3.3.2. Algorithm Leafify

Algorithm Leafify takes as input a DNSTP solution and the set of terminals and returns a DNSTP solution in which terminals are leaves. To construct the new DNSTP solution $\hat{\mathcal{T}}$ in which terminals are leaves, the Algorithm Leafify initially computes the terminal connection subgraph $G_{\mathcal{T}}^{\vec{TS}}$ and computes a maximal mathching $\mathcal{M}$ on it (see Lines 2,3). In the loop from Lines 4-8 each terminal is reconnected, such that only the connection to Steiner node s_0, to which it is connected in $\mathcal{M}$, is preserved. In exchange, Steiner nodes are connected in a line.

Algorithm 3.2: Leafify

Input : DNSTP solution $\mathcal{T}$ on $G_R^{SP} = (V_G^{SP}, E_G^{SP})$, Set of Terminals T

Output: DNSTP solution $\hat{\mathcal{T}}$ on $G_R^{SP} = (V_G^{SP}, E_G^{SP})$ with $\delta_{\hat{T}}(t) = 1$ for all $t \in T$

1 **set** $\hat{\mathcal{T}} \triangleq \mathcal{T}$
2 **construct** $G_{\mathcal{T}}^{\vec{TS}} = (V_{\mathcal{T}}^{\vec{TS}}, E_{\mathcal{T}}^{\vec{TS}})$ **from** $\mathcal{T}$
3 **compute maximal matching** $\mathcal{M} \subset E_{\mathcal{T}}^{\vec{TS}}$ on $G_{\mathcal{T}}^{\vec{TS}}$
4 **foreach** $t \in V_{\mathcal{T}}^{\vec{TS}}$ **do**
5 **let** $s_0 \in V_{\mathcal{T}}^{\vec{TS}}$ **, such that** $\{t, s_0\} \in \mathcal{M}$
6 **set** $\bar{S} \triangleq \{s_1, s_2, s_3, \ldots, s_n | 1 \leqslant i \leqslant n, \{t, s_i\} \in E_{\mathcal{T}}^{\vec{TS}}\}$
7 **set** $\hat{\mathcal{T}} \leftarrow (\hat{\mathcal{T}} \backslash \{\{t, \bar{s}\} | \bar{s} \in \bar{S}\}) \cup \{\{s_{i-1}, s_i\} | 1 \leqslant i \leqslant n\}$
8 **end**
9 **return** $\hat{\mathcal{T}}$

Note that by Lemma 3.29 there exists for the computed matching $|\mathcal{M}| = |V_{\mathcal{T}}^{\vec{TS}}|$ holds and therefore the algorithm is well defined, as $s_0 \in V_{\mathcal{T}}^{\vec{TS}}$ as chosen in Line 5 must always exist. We further note that the runtime of Algorithm Leafify is dominated by computing a maximal matching, which can be obtained on bipartite graphs in polynomial time [KV12].

3.3.3. Proof of Correctness of Algorithm Leafify

We only shortly prove the correctness of ALgorithm Leafify.

Lemma 3.30: *(Algorithm Leafify is correct.)* We prove that given a feasible DNSTP solution, i.e. a Steiner tree $\mathcal{T}$, Algorithm Leafify outputs a feasible Steiner tree $\hat{\mathcal{T}}$ with $\delta_{\hat{\mathcal{T}}}(t) = 1$ for all $t \in T$.

Proof: Based on the connection scheme employed in Line 7 of Algorithm Leafify, connectivity is preserved in each iteration of the loop. Therefore $\hat{\mathcal{T}}$ is still connected. To check that $\hat{\mathcal{T}}$ is indeed a tree, note that cycles within $\hat{\mathcal{T}}$ directly translate to cycles in $G_{\mathcal{T}}^{\vec{TS}}$, which is impossible. Lastly, as all but one edge that did connect $t \in T$ to $V_{\mathcal{T}}^{\vec{TS}} \backslash T$ is removed, the degree of all terminals is one. ■

The following theorem was already proven in a similar form by Fuchs [Fuc03].

Theorem 3.31: ***Algorithm Leafify increases the cost by at most a factor of 2.***

Given a feasible DNSTP solution $\mathcal{T}$ on G_R^{SP} which might violate degree constraints of terminals T, for the DNSTP solution $\hat{E}_{\mathcal{T}}$ obtained by calling Algorithm Leafify holds

$$C_{\text{DNSTP}}(\hat{\mathcal{T}}) \leqslant 2C_{\text{DNSTP}}(\mathcal{T}) \,.$$

Proof: We show that the edge costs are doubled at most, as costs for opening Steiner sites are not changed. For each newly introduced edge $\{s_{i-1}, s_i\}$ (see Line 7), the costs can be bounded by the triangle inequality on G_R^{SP} (see Observation 3.4) by $c_E(\{s_{i-1}, s_i\}) \leqslant c_E(\{t, s_{i-1}\}) + c_E(\{t, s_i\})$. Therefore, the sum of costs for newly introduced edges is

$$c_E(\{t, s_0\}) + 2 \cdot \sum_{i=1}^{n-1} c_E(\{t, s_i\}) + c_E(\{t, s_n\})$$

As all edges $\{t, s_i\}$ for $1 \leqslant n$ are removed, the edge costs double at most, concluding the proof of the theorem. ■

Theorem 3.32: ***Algorithm Leafify increases the degree by at most a factor of 2.***

Given a feasible DNSTP solution $\mathcal{T}$ on G_R^{SP} which might violate degree constraints of terminals T, and having obtained the DNSTP solution $\hat{\mathcal{T}}$ by calling Algorithm Leafify, for each non-terminal node $v \in \hat{\mathcal{T}} \setminus T$ holds

$$\delta_{\hat{\mathcal{T}}}(v) \leqslant 2\delta_{\mathcal{T}}(v).$$

Proof: Based on the swapping of edges in Line 7, the degree of nodes $\{s_0, s_1, \ldots, s_{n-1}\}$ increases by one. Note however, that terminal t connected to all of the above Steiner nodes before. Therefore, in each iteration of the loop in Lines 4-8 the degree of Steiner nodes is increased at most by one *per each connected terminal*. Therefore, the degree of each Steiner node (or of the root) is at most doubled. ■

3.3.4. Logarithmic Bi-Criteria Approximation of NVSTP

Having proven the correctness of Algorithm Leafify in the above section, we can now prove Theorem 3.22.

Proof (Theorem 3.22)**:** The following algorithm yields a bi-criteria $\mathcal{O}(\log |T|)$-approximation for any NVSTP instance. The algorithmic outline described in Observation 3.25 is adapted in such a way, that we require $u_V^{SP}(t) \triangleq 2$ for all terminals $t \in T$ and that we compute an approximate solution $\mathcal{T}$ to the corresponding DNSTP instance on G_R^{SP} according to Theorem 3.21. Then Algorithm Leafify is used to ensure that terminals in the Steiner tree $\hat{\mathcal{T}}$ are leaves. By introducing $\pi(\{u, v\}) \triangleq \{u, v\}$ for $\{u, v\} \in \hat{\mathcal{T}}$, $\hat{\mathcal{T}}$ is a virtual tree for the NVSTP instance on G_R^{SP} which by 3.24 defines a virtual tree on the original instance.

Based on Theorems 3.31 and 3.32 both the cost and the degree violations are only increased by a factor of two. Since DNSTP is a generalization of NVSTP, as terminals are not constrained to be leaves, this proves that the Algorithm described indeed yields a logarithmic bi-criteria approximation of NVSTP, as the runtime of Algorithm Leafify is polynomial. ■

Part II.

Exact Algorithms for CVSAP

4. A Multi-Commodity Flow Formulation ★

This section introduces a naive multi-commodity flow (MCF) formulation (see A-CVSAP-MCF) to solve (A-)CVSAP. The formulation A-CVSAP-MCF models the virtual arborescence searched for rather directly, as it uniquely determines virtual links and paths for active Steiner nodes. This explicit representation comes at the price of a substantially larger model. In Section 14 we provide a computational comparison showing the superiority of our compact formulation developed in Section 5.

4.1. Notation

For ease of representation of A-CVSAP-MCF we use an extended graph similar to the one introduced in Section 5.

Definition 4.1: EXTENDED GRAPH FOR A-CVSAP-MCF Given a directed network $G = (V_G, E_G, c_E, u_E)$ and a request $R_G = (r, S, T, u_r, c_S, u_S)$ as introduced in Section 2.2, we define the *extended graph* $G_{\text{MCF}} = (V_{\text{MCF}}, E_{\text{MCF}})$ for the A-CVSAP-MCF formulation as follows:

(EXT-1-MCF) $V_{\text{MCF}} \triangleq V_G \cup \{\mathrm{o}^-\}$,

(EXT-2-MCF) $E_{\text{MCF}} \triangleq E_G \cup \{(r, \mathrm{o}^-)\} \cup E^{S^-}_{\text{MCF}}$,

where $E^{S^-}_{\text{MCF}} \triangleq S \times \{\mathrm{o}^-\}$. □

We use the Kronecker-Delta $\delta_{\mathsf{x},\mathsf{y}} \in \{0, 1\}$, where $\delta_{\mathsf{x},\mathsf{y}} = 1$ holds iff. $\mathsf{x} = \mathsf{y}$. Flow variables corresponding to different commodities are distinguished by superscripts and we use $f^{\mathsf{x}}(\mathsf{Y})$ to denote $\sum_{y \in \mathsf{Y}} f^{\mathsf{x}}(\mathsf{y})$. We denote the set of feasible solution for A-CVSAP-MCF by $\mathcal{F}_{\text{MCF}}$.

4.2. The MIP Model

The formulation A-CVSAP-MCF uses one commodity for each Steiner site (see MCF-10) and a single commodity for the flow originating at the terminals (see MCF-9). Note that while f^s defines a flow variable for each Steiner site $s \in S$ we use f^T to denote a single commodity for all terminals. Furthermore note that flow variables f^s corresponding to Steiner sites are binary whereas the aggregated flow variables f^T from the terminals are defined to be integers.

Integer Program 4.1: A-CVSAP-MCF

$$
\begin{aligned}
\text{minimize} \quad & C_{\text{MCF}} = \sum_{e \in E_G} \mathbf{c}_e (f_e + \sum_{s \in S} f_{s,e}) && && \text{(MCF-OBJ)} \\
& \qquad + \sum_{s \in S} \mathbf{c}_s \cdot x_s \\
\text{subject to} \quad & f^T(\delta^+_{E_{\text{MCF}}}(v)) = f^T(\delta^-_{E_{\text{MCF}}}(v)) + |\{v\} \cap T| && \forall\, v \in V_G && \text{(MCF-1)} \\
& f^s(\delta^+_{E^S_{\text{MCF}}}(v)) = f^s(\delta^-_{E^S_{\text{MCF}}}(v)) + \delta_{s,v} \cdot x_s && \forall\, s \in S, v \in V_G && \text{(MCF-2)} \\
& f^T_e + \sum_{s \in S} f^s_e \leqslant \begin{cases} \mathbf{u}_s x_s, & e = (s, \text{o}^-), s \in S \\ \mathbf{u}_r \quad , & e = (r, \text{o}^-) \\ \mathbf{u}_e \quad , & e \in E_G \end{cases} && \forall e \in E_{\text{MCF}} && \text{(MCF-3)} \\
& -|S|(1 - f^s_{\bar{s},\text{o}^-}) \leqslant p_s - p_{\bar{s}} - 1 && \forall\, s, \bar{s} \in S && \text{(MCF-4)} \\
& f^s_{(\bar{s},\text{o}^-)} \leqslant x_{\bar{s}} && \forall\, s \in S, \bar{s} \in S - s && \text{(MCF-5}^\star\text{)} \\
& f^s_{s,\text{o}^-} = 0 && \forall\, s \in S && \text{(MCF-6}^\star\text{)} \\
& f^s_{\bar{s},\text{o}^-} + f^{\bar{s}}_{s,\text{o}^-} \leqslant 1 && \forall\, s, \bar{s} \in S && \text{(MCF-7}^\star\text{)} \\
& x_s \in \{0,1\} && \forall\, s \in S && \text{(MCF-8)} \\
& f^T_e \in \mathbb{Z}_{\geqslant 0} && \forall\, e \in E_{\text{MCF}} && \text{(MCF-9)} \\
& f^s_e \in \{0,1\} && \forall\, s \in S, e \in E_{\text{MCF}} && \text{(MCF-10)} \\
& p \in [0, |S| - 1] && \forall\, s \in S && \text{(MCF-11)}
\end{aligned}
$$

We now briefly describe how a solution $(x, p, f^s, f^T) \in \mathcal{F}_{\text{MCF}}$ relates to a virtual arborescence $\hat{\mathcal{T}}_G = (\hat{V}_{\mathcal{T}}, \hat{E}_{\mathcal{T}}, \hat{r}, \hat{\pi}) \in \mathcal{F}_{\text{A-CVSAP}}$. We naturally set $\hat{V}_{\mathcal{T}} \triangleq \{r\} \cup \{s \in S | \hat{x}_s \geqslant 1\} \cup T$ and $\hat{r} = r$. We continue by showing how $\hat{E}_{\mathcal{T}}$ and $\hat{\pi}$ can be retrieved.

Constraints MCF-1 and MCF-2 specify flow preservation for the commodities such that terminal nodes emit one unit of flow in f^T and activated Steiner nodes emit one unit of flow in f^s. Note that in Constraint MCF-2 $\delta_{s,v}$ is a constant. As these constraints are specified for nodes $v \in V_G$, flows in f^T and f^s must terminate in o^- via edges in $E^{S^-}_{\text{MCF}}$ or via (r, o^-).

If a Steiner node $s \in S$ is activated, f^s defines a path P^s from s to o^-. We therefore include $e = (s, P^s_{|P^s|-1})$ in $\hat{E}_{\mathcal{T}}$ and set $\hat{\pi}(e) = \langle P^s_1, \ldots, P^s_{|P^s|-1} \rangle$. As we use a single commodity for flow originating at the terminals, we have to first decompose f^T into paths $\{P^t | t \in T\}$ such that P^t originates at t and terminates in o^-. Due to the single destination, this can always be done using the standard $s - t$ flow decomposition [AMO93].

As the capacity constraints MCF-3 effectively constrain the number of incoming connections and the validity of edge capacities, we only need to establish the validity of connectivity condition VA-2 to show that $\hat{\mathcal{T}}_G \in \mathcal{F}_{\text{A-CVSAP}}$ holds. As terminals and active Steiner nodes must be connected as discussed above, VA-2 may only be violated by $\hat{\mathcal{T}}_G$ if a cycle exists

in $\hat{E}_{\mathcal{T}}$. To forbid such cycles, we adapt the well-known Miller-Tucker-Zemlim (MTZ) constraints [CCL09] using continuous priority variables $p_s \in [0, |S| - 1]$ in MCF-4. The MTZ constraint MCF-4 enforces $f^s(\bar{s}, \mathrm{o}^-) = 1 \Rightarrow p_s \geqslant p_{\bar{s}} + 1$, forbidding cyclic assignments containing only Steiner nodes. As terminals may not receive flow and the root may not send flow, this suffices to forbid cycles in $\hat{E}_{\mathcal{T}}$ overall and thus $\hat{\mathcal{T}}_G \in \mathcal{F}_{\text{A-CVSAP}}$ holds.

As formulations relying on MTZ constraints are comparatively weak [PVD01], we introduce additional valid inequalities MCF-5*, MCF-6* and MCF-7* to strengthen the formulation. Constraint MCF-6* disallows Steiner node $s \in S$ to absorb its own flow and MCF-7* explicitly forbids cycles of length 2. Lastly, Constraint MCF-5* forces Steiner nodes receiving flow from another Steiner node to be activated.

4.3. Implementation

We have implemented the IP formulation A-CVSAP-MCF in the GNU Mathematical Programming Language (GMPL) that is part of the GNU Linear Programming Kit (GLPK) [GNU13]. Given a GMPL model file, GLPK can be used to produce an .lp file which can be solved using commercial solvers as CPLEX [CPL13]. The model and the data files can be obtained from [RS13a].

5. VirtuCast Algorithm

In this section we present the Algorithm VirtuCast to solve CVSAP. VirtuCast first computes a solution for a single-commodity flow Integer Programming formulation and then constructs the corresponding Virtual Arborescence. Even though our IP formulation can be used to compute the optimal solution for any CVSAP instance, feasible solutions to our IP formulation already yield feasible solutions to CVSAP. This allows to derive near-optimal solutions *during* the solution process.

5.1. The IP Model

Our IP (see IP-A-CVSAP) is based on an *extended graph* containing a single super source o^+ and two distinct super sinks o_S^- and o_r^- (see Definition 5.1). While o_r^- may only receive flow from the root r, all possible Steiner sites $s \in S$ connect to o_S^-. Distinguishing between these two super sinks is necessary, as we will require activated Steiner nodes to not *absorb all* incoming flow, but forward at least one unit of flow towards o_r^-, which will indeed ensure connectivity.

Definition 5.1: EXTENDED GRAPH Given a directed network $G = (V_G, E_G, c_E, u_E)$ and a request $R_G = (r, S, T, u_r, c_S, u_S)$ as introduced in Section 2.2 we define the *extended graph* $G_{\text{ext}} = (V_{\text{ext}}, E_{\text{ext}})$ as follows

(EXT-1) $V_{\text{ext}} \triangleq V_G \cup \{o^+, o_S^-, o_r^-\}$,

(EXT-2) $E_{\text{ext}} \triangleq E_G \cup \{(r, o_r^-)\} \cup E_{\text{ext}}^{S^-} \cup E_{\text{ext}}^{S^+} \cup E_{\text{ext}}^{T^+}$,

where $E_{\text{ext}}^{S^-} \triangleq S \times \{o_S^-\}$, $E_{\text{ext}}^{S^+} \triangleq \{o^+\} \times S$ and $E_{\text{ext}}^{T^+} \triangleq \{o^+\} \times T$. We define $E_{\text{ext}}^{R} \triangleq E_{\text{ext}} \backslash E_{\text{ext}}^{S^-}$. □

Further Notation.

To clearly distinguish between variables and constants, we typeset constants in bold font: instead of referring to c_E, c_S and u_E, u_r, u_S we use $\mathbf{c}_y$ and $\mathbf{u}_y$, where y may either refer to an edge or a Steiner site. Similarly, we use $\mathbf{u}_y$ where y may either refer to an edge, the root or Steiner node. We abbreviate $\sum_{y \in Y} f_y$ by $f(Y)$. We use $Y + y$ to denote $Y \cup \{y\}$ and $Y - y$ to denote $Y \backslash \{y\}$ for a set Y and a singleton y. For $f \in \mathbb{Z}_{\geqslant 0}^{E_{\text{ext}}}$ we define the flow-carrying subgraph $G_{\text{ext}}^f \triangleq (V_{\text{ext}}^f, V_{\text{ext}}^f)$ with $V_{\text{ext}}^f \triangleq V_{\text{ext}}$ and $V_{\text{ext}}^f \triangleq \{e | e \in E_{\text{ext}} \wedge f(e) \geqslant 1\}$.

The IP formulation IP-A-CVSAP uses an integral single-commodity flow and we define a flow variable $f_e \in \mathbb{Z}_{\geqslant 0}$ for each edge $e \in E_{\text{ext}}$ in the extended graph (see IP-11). As we use an aggregated flow formulation, that does not model routing decisions explicitly, we show in Section 5.2 how this single-commodity flow can be decomposed into paths for constructing an actual solution for CVSAP.

Whether a Steiner site $s \in S$ is activated is decided by the binary variable $x_s \in \{0, 1\}$ (see IP-10). Constraint IP-8 forces each terminal $t \in T$ to send a single unit of flow. As flow conservation is enforced on all original nodes $v \in V_G$ (see IP-1), all flow originating at o^+ must be forwarded to one of the super sinks o_r^- or o_S^-, while not violating link capacities (see IP-7).

Integer Program 5.1: IP-A-CVSAP

$$\text{minimize} \quad C_{\text{IP}}(x, f) = \sum_{e \in E_G} \mathbf{c}_e f_e + \sum_{s \in S} \mathbf{c}_s x_s \tag{IP-OBJ}$$

$$\text{subject to} \quad f(\delta^+_{E_{\text{ext}}}(v)) = f(\delta^-_{E_{\text{ext}}}(v)) \qquad \forall\, v \in V_G \tag{IP-1}$$

$$f(\delta^+_{E^R_{\text{ext}}}(W)) \geqslant x_s \qquad \forall\, W \subseteq V_G, s \in W \cap S \neq \varnothing \tag{IP-2}$$

$$f(\delta^+_{E^R_{\text{ext}}}(W)) \geqslant 1 \qquad \forall\, W \subseteq V_G, T \cap W \neq \varnothing \tag{IP-3*}$$

$$f_e \geqslant x_s \qquad \forall\, e = (s, \mathrm{o}_S^-) \in E^{S^-}_{\text{ext}} \tag{IP-4*}$$

$$f_e \leqslant \mathbf{u}_s x_s \qquad \forall\, e = (s, \mathrm{o}_S^-) \in E^{S^-}_{\text{ext}} \tag{IP-5}$$

$$f_{(r, \mathrm{o}_r^-)} \leqslant \mathbf{u}_r \tag{IP-6}$$

$$f_e \leqslant \mathbf{u}_e \qquad \forall\, e \in E_G \tag{IP-7}$$

$$f_e = 1 \qquad \forall\, e \in E^{T^+}_{\text{ext}} \tag{IP-8}$$

$$f_e = x_s \qquad \forall\, e = (\mathrm{o}^+, s) \in E^{S^+}_{\text{ext}} \tag{IP-9}$$

$$x_s \in \{0, 1\} \qquad \forall\, s \in S \tag{IP-10}$$

$$f_e \in \mathbb{Z}_{\geqslant 0} \qquad \forall\, e \in E_{\text{ext}} \tag{IP-11}$$

As the definition of A-CVSAP requires that each terminal $t \in T$ establishes a path to r, we need to enforce connectivity; otherwise active Steiner nodes would simply absorb flow by directing it towards o_S^-. To prohibit this, we adopt well-known *Connectivity Inequalities* [LR04] and *Directed Steiner Cuts* [KM98]. Our Connectivity Inequalities IP-2 state that each set of nodes containing a Steiner site $s \in S$ must emit at least one unit of flow in E^R_{ext}, if s is activated. As E^R_{ext} does not contain edges towards o_S^-, this constraint therefore enforces that there exists a path in G^f_{ext} from each activated Steiner node s to the root r.

Analogously, the Directed Steiner Cuts IP-3* enforce that there exists a path from each terminal $t \in T$ towards r in G^f_{ext}. These directed Steiner cuts constitute valid inequalities which are implied by IP-1 and IP-2 (see Lemma 5.4). These Directed Steiner Cuts can strengthen the model by improving the LP relaxation during the branch-and-cut process, as the next lemma

shows. As they are not needed for proving the correctness and could technically be removed, we mark them with a * (star).

Lemma 5.2: The directed Steiner cuts (see IP-3*) can strengthen the formulation 5.1, i.e. improve the objective value of its LP relaxation.

Proof: Consider the simple example in which the whole network $G = (V_G, E_G)$ consists only of three nodes on a line: $V_G = \{r, s, t\}$ and $E_G = \{(t, s), (s, r)\}$. We consider the following capacities and costs $u_S(s) = 10$, $u_r(r) = 1$, $c_E(t, s) = c_E(s, r) = 1$, $c_S(s) = 5$ and that r is the root, s is the single Steiner location and t the only terminal. The optimal solution of 5.1 without IP-3* and relaxing the constraints IP-11 and IP-10 to $f \in \mathbb{R}_{\geqslant 0}^{E_{\text{ext}}}$ and $x \in [0, 1]^S$ is $f(t, s) = 1$, $x_s = 1/10$ and $f(s, r) = 1/10$ yielding an objective value of $5 \cdot x_s + f(t, s) + f(s, r) = 1.6$ By introducing Constraint IP-3* $f(s, r)$ must equal 1 and therefore, the solution obtained by introducing this constraint yields the integral solution $f(t, s) = f(s, r) = 1$ and $x_s = 0$ with objective value 2, therefore strengthening the model. Note that we did not give values for the edges from and to the super sinks which are introduced in E_{ext} as these do not influence the objective value. ■

As a Steiner node $s \in S$ is activated iff. $x_s = 1$, Constraint IP-9 requires activated Steiner nodes to receive one unit of flow while being able to maximally absorb $\mathbf{u}_s$ many units of flow by forwarding it to o_S^- (see IP-5). Furthermore, by IP-5 inactive Steiner sites may not absorb flow at all. The Constraint IP-4* requires active Steiner nodes to at least absorb one unit of flow. This is a valid inequality, as activating a Steiner site $s \in S$ incurs non-negative costs. We introduce this constraint here, as it specifies a condition that is used in a proof later on. Constraint IP-6 defines an upper bound on the amount of flow that the root may receive and the objective function IP-OBJ mirrors the CVSAP cost function (see Definition 2.3). We denote with $\mathcal{F}_{\text{IP}} = \{(x, f) \in \{0, 1\}^S \times \mathbb{Z}_{\geqslant 0}^{E_{\text{ext}}} | \text{IP-1 - IP-11}\}$ the set of feasible solutions to IP-A-CVSAP and with $\mathcal{F}_{\text{LP}}$ the solution space of IP-A-CVSAP in which the variables are relaxed to $x_s \in [0, 1]$ and $f_e \in \mathbb{R}_{\geqslant 0}$.

5.2. Flow Decomposition

Given a feasible solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$ for IP-A-CVSAP, Algorithm Decompose constructs a feasible solution $\hat{\mathcal{T}}_G \in \mathcal{F}_{\text{A-CVSAP}}$ for CVSAP. Similarly to well-known algorithms for computing flow decompositions for simple s-t flows (see e.g. [AMO93]), our algorithm iteratively deconstructs the flow into paths from the super source o^+ to the super sinks o_S^- or o_r^-, which are successively removed from the network. However, as IP-A-CVSAP does not pose a simple flow problem, we constantly need to ensure that Connectivity Inequalities IP-2 hold after removing flow in $G_{\text{ext}}^{\hat{f}}$. We first present Decompose in more detail and prove its correctness. A short runtime analysis is contained in Section 5.3.

5.2.1. Synopsis of Algorithm.

Algorithm Decompose constructs a feasible VA $\hat{\mathcal{T}}_G$ given a solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$. In Line 2, $\hat{\mathcal{T}}_G$ is initialized without any edges but containing all the nodes the final solution will consist of, namely the root r, the terminals T and the activated Steiner nodes $\{s \in S | x_s \geqslant 1\}$. In Line 3 a terminal node $t \in \hat{T}$ is selected for which a path is constructed to either an active Steiner node or to the root itself (Lines 5-13). In Line 5 a path P, connecting t to the root r in the flow network $G^{\hat{f}}_{\text{ext}}$, is chosen (see Lemma 5.4 for the proof of existence for such a path). Note that by definition of $G^{\hat{f}}_{\text{ext}}$ all edges contained in P carry at least one unit of flow. Within the loop beginning in Line 6, the flow on path P is iteratively decremented (see Line 7) as long as the Connectivity Inequality IP-2 is not violated. In case it is violated, we revert the reduction of flow (see Line 11) and select a path towards the super sink o^-_S starting at the current node P_j (see Line 10). Such a path must exist according to Lemma 5.6. The path P is accordingly redirected in Line 12 .

The path construction (in Lines 5 to 12) terminates once the flow from the second last node $P_{|P|-1}$ towards the last node $P_{|P|}$ has been reduced. By construction, the path P leads from the super source o^+ via the terminal $t \in \hat{T}$ towards the super sink o^-_r or o^-_S. If P terminates in o^-_S via Steiner node $s = P_{|P|-1} \in \hat{S}$ such that (s, o^-_S) carries no flow anymore, s itself becomes a terminal (see Lines 15 and 16). Otherwise, P terminates in o^-_r and $P_{|P|-1} = r$ holds. Lastly, in Line 18 the (virtual) edge $(t, P_{|P|-1})$ is added to $\hat{E}_{\mathcal{T}}$ and $\hat{\pi}(t, P_{|P|-1})$ is set accordingly to the truncated path P, where head, tail and any cycles are removed (function `simplify`).

5.2.2. Proof of Correctness

We will now formally prove the correctness of Algorithm Decompose, thereby showing that IP-A-CVSAP can be used to compute (optimal) solutions to CVSAP. We use an inductive argument similar to the one used for proving the existence of flow decompositions (see [AMO93]). we assume that all constraints of IP-A-CVSAP hold and show that for any terminal $t \in T$ a path towards the root or to an active Steiner node can be constructed, such that decrementing the flow along the path by one unit does again yield a feasible solution to IP-A-CVSAP, in which t has been removed from the set of terminals (see Theorem 5.3 below). During the course of this induction, the well-definedness of the **choose** operations is shown.

Theorem 5.3: ***Induction Step***

> *Assuming that the constraints of Decompose hold with respect to $\hat{S}, \hat{T}, \hat{f}, \hat{x}$ before executing Line 4, then the constraints of Decompose will also hold in Line 18 with respect to then reduced problem $\hat{S}, \hat{T}, \hat{f}, \hat{x}$.*

To prove the above theorem, we use the following Lemmas 5.4 through 5.6.

Lemma 5.4: Assuming that IP-1 and IP-2 hold, there exists a path $P = \langle \text{o}^+, t, \ldots, \text{o}^-_r \rangle \in G^{\hat{f}}_{\text{ext}}$ in Line 5.

Algorithm 5.1: Decompose

Input : Network $G = (V_G, E_G, c_E, u_E)$, Request $R_G = (r, S, T, u_r, c_S, u_S)$, Solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$ to IP-A-CVSAP

Output: Feasible Virtual Arborescence $\hat{\mathcal{T}}_G$ for CVSAP

1 **set** $\hat{S} \triangleq \{s \in S | x_s \geqslant 1\}$ **and** $\hat{T} \triangleq T$
2 **set** $\hat{\mathcal{T}}_G \triangleq (\hat{V}_{\mathcal{T}}, \hat{E}_{\mathcal{T}}, r, \hat{\pi})$ **where** $\hat{V}_{\mathcal{T}} \triangleq \{r\} \cup \hat{S} \cup \hat{T}$, $\hat{E}_{\mathcal{T}} \triangleq \varnothing$ **and** $\hat{\pi} : \hat{E}_{\mathcal{T}} \to \mathcal{P}_G$
3 **while** $\hat{T} \neq \varnothing$ **do**
4 **let** $t \in \hat{T}$ **and** $\hat{T} \leftarrow \hat{T} - t$
5 **choose** $P \triangleq \langle \mathrm{o}^+, t, \ldots, \mathrm{o}_r^- \rangle \in G_{\text{ext}}^{\hat{f}}$
6 **for** $j = 1$ **to** $|P| - 1$ **do**
7 **set** $\hat{f}(P_j, P_{j+1}) \leftarrow \hat{f}(P_j, P_{j+1}) - 1$
8 **if** Constraint IP-2 is violated *with respect to* $\hat{f}$ and $\hat{S}$ **then**
9 **choose** $W \subseteq V_G$ **such that** $W \cap \hat{S} \neq \varnothing$ **and** $\hat{f}(\delta^+_{E^R_{\text{ext}}}(W)) = 0$
10 **choose** $P' \triangleq \langle P_j, \ldots, \mathrm{o}_S^- \rangle \in G_{\text{ext}}^{\hat{f}}$ **such that** $P_i \in W$ for $1 \leqslant i < m$
11 **set** $\hat{f}(P_j, P_{j+1}) \leftarrow \hat{f}(P_j, P_{j+1}) + 1$ **and** $\hat{f}(P'_1, P'_2) \leftarrow \hat{f}(P'_1, P'_2) - 1$
12 **set** $P \leftarrow \langle P_1, \ldots, P_{j-1}, P_j = P'_1, P'_2, \ldots, P'_m \rangle$
13 **end**
14 **end**
15 **if** $P_{|P|} = \mathrm{o}_S^-$ **and** $\hat{f}(P_{|P|-1}, P_{|P|}) = 0$ **then**
16 **set** $\hat{S} \leftarrow \hat{S} - P_{|P|-1}$ **and** $\hat{x}(P_{|P|-1}) \leftarrow 0$ **and** $\hat{T} \leftarrow \hat{T} + P_{|P|-1}$
17 **end**
18 **set** $\hat{E}_{\mathcal{T}} \leftarrow \hat{E}_{\mathcal{T}} + (t, P_{|P|-1})$ **and** $\hat{\pi}(t, P_{|P|-1}) \triangleq \texttt{simplify}(\langle P_2, \ldots, P_{|P|-1} \rangle)$
19 **end**
20 **return** $\hat{\mathcal{T}}_G$

Proof: Note that initially (i.e. in Line 1) $\hat{f}(\mathrm{o}^+, v) = 1$ holds for $v \in \hat{S} \cup \hat{T}$ by IP-8 and IP-9. This flow will only be reduced once, as a node $t \in \hat{T}$ will only be handled once when it is removed from $\hat{T}$ in Line 4, and similarly, a node $s \in \hat{S}$ will only be moved once into $\hat{T}$ in Line 16. By flow conservation (see IP-1), there must exist a path from t to either o_r^- or o_S^-. However, as we assume IP-2 to hold, there exists a path from each $s \in \hat{S}$ to o_r^- and we conclude that such a path $P = \langle \mathrm{o}^+, t, \ldots, \mathrm{o}_r^- \rangle \in G_{\text{ext}}^{\hat{f}}$ must exist. ■

Lemma 5.5: Assuming that IP-1 has held in Line 5, $f(\delta^+_{E_{\text{ext}}}(v)) - f(\delta^-_{E_{\text{ext}}}(v)) = \delta_{v,P_{j+1}}$ holds for all $v \in V_G$ during construction of P (Lines 8-13), where $\delta_{x,y} \in \{0, 1\}$ and $\delta_{x,y} = 1$ iff. $x = y$.

Proof: We prove this statement by an inductive argument assuming for now that **choose** operations in Lines 9 and 10 are well-defined.

After the first execution of Line 7, $f(\delta^+_{E_{\text{ext}}}(P_2 = t)) - f(\delta^-_{E_{\text{ext}}}(P_2 = t)) = 1$ holds, while for no other node $v \in V_G$ flow on adjacent edges were changed, and therefore $f(\delta^+_{E_{\text{ext}}}(v)) -$

$f(\delta^-_{E_{\text{ext}}}(v)) = 1$ holds. Furthermore, the reduction of flow on edge $(\text{o}^+, P_2 = t)$ cannot violate IP-2, such that our claim holds until Line 13 and therefore for the base case $j = 1$.

Assuming that $f(\delta^+_{E_{\text{ext}}}(v)) - f(\delta^-_{E_{\text{ext}}}(v)) = \delta_{v,P_{j+1}}$ has held for $j = n$, it is easy to check that it will continue to hold for $j' = n + 1$, as either in Line 7 or in Line 11 the outgoing flow from node $P_{j'}$ towards node $P_{j'+1}$ is reduced such that $f(\delta^+_{E_{\text{ext}}}(v)) - f(\delta^-_{E_{\text{ext}}}(v)) = \delta_{v,P_{j'+1}}$ indeed holds for all $v \in V_G$. ■

Lemma 5.6: Assuming that connectivity inequalities IP-2 have held before executing Line 7, these inequalities will hold again at Line 13.

Proof: We only have to consider the case in which the Constraint IP-2 was violated after executing Line 7. Assume therefore that IP-2 is violated in Line 8. The **choose** operation in Line 9 is well-defined, as IP-2 is violated. Let $W \subseteq V_G$ be any violated set with $\hat{S} \cap W \neq \varnothing$. To prove this lemma, we prove the following four statements:

(a) P_j is contained in W while P_{j+1} is not contained in W.

(b) $\hat{f}(P_j, P_{j+1}) = 0$ holds in Lines 9-10.

(c) Before flow reduction in Line 7, there existed a path $P'' = \langle s, \ldots, P_j, P_{j+1}, \ldots, \text{o}^-_r \rangle \in G^{\hat{f}}_{\text{ext}}$ for $s \in \hat{S} \cap W$.

(d) There exists a path $P' = \langle P_j, \ldots, \text{o}^-_S \rangle$ with $P'_i \in W$ for $1 \leqslant i < |P'|$ in $G^{\hat{f}}_{\text{ext}}$.

Considering (a), note that edge (P_j, P_{j+1}) is by definition only included in $\delta^+_{E_{\text{ext}}}(W)$ if $P_j \in W$ and $P_j \notin W$. Thus, assuming that either P_j is not contained in W or assuming that P_{j+1} is contained in W, we can conclude that edge (P_j, P_{j+1}) is not contained in $\delta^+_{E_{\text{ext}}}(W)$. However, in this case the connectivity inequality IP-2 must have been violated even before flow was reduced. This contradicts our assumption that connectivity inequalities IP-2 have held beforehand, therefore proving (a).

The correctness of (b) directly follows from (a), as by (a) $(P_j, P_{j+1}) \in \delta^+_{E_{\text{ext}}}(W)$ holds. As $\hat{f}(\delta^+_{E_{\text{ext}}}(W)) = 0$ holds by definition of W and flow may not be negative, we derive the second statement.

We now prove the statement (c). As connectivity inequalities IP-2 are assumed to have held *before* the flow reduction in Line 7, for each activated Steiner node $s \in \hat{S}$ there existed a path from s to o^-_r in $G^{\hat{f}}_{\text{ext}}$. By the second statement, (P_j, P_{j+1}) is the only edge in $G^{\hat{f}}_{\text{ext}}$ leaving W showing that indeed a path $P'' = \langle s, \ldots, v, P_j, P_{j+1}, \ldots, \text{o}^-_r \rangle \in G^{\hat{f}}_{\text{ext}}$ for $s \in \hat{S}$ existed *before* reduction of flow on (P_j, P_{j+1}).

By statement (c), the prefix $\langle s, \ldots, P_j \rangle$ of path P'' still exists in $G^{\hat{f}}_{\text{ext}}$ inducing that P_j is reached by a positive flow. By Lemma 5.5 flow conservation holds for all nodes $w \in W$, since by statement (a) P_{j+1} is not included in W. As o^-_r is not included in W, there must exist a path $P' = \langle P_j, \ldots, \text{o}^-_S \rangle \in G^{\hat{f}}_{\text{ext}}$ with $P_i \in W$ for $1 \leqslant i < m$. This shows the fourth statement (d) and shows that the **choose** operation in Line 10 is well-defined.

We will now prove the main statement of this lemma, namely that in Line 13 the connectivity inequalities IP-2 hold (again). In Line 11, the flow along edge (P_j, P_{j+1}) is incremented

again. Assume for the sake of contradiction, that the reduction of flow along (P'_1, P'_2) violates a connectivity inequality with node set W' such that $\hat{f}(\delta^+_{E_{\text{ext}}}(W')) = 0$ holds. By the same argument as used for proving statement (a), it is easy to see that $P'_1 \in W'$ and $P'_2 \notin W'$ must hold. However, by statement (c), after having reverted the flow reduction along (P_j, P_{j+1}), the path $\langle P_j, P_{j+1}, \ldots, o_r^- \rangle$ was re-established in $G^{\hat{f}}_{\text{ext}}$. As flow along any of the edges contained in this path is greater or equal to one, W' cannot possibly violate IP-2 and contain $P_j \in W'$ as the super sink for the root $o_r^- \notin W' \subseteq V_G$ may never be contained in W'. ■

Using the above lemma, we can now prove Theorem 5.3.

Proof (Theorem 5.3)**:** Assume that the constraints of IP-A-CVSAP hold with respect to $\hat{S}, \hat{T}, \hat{f}, \hat{x}$ before executing Line 4. By Lemma 5.4 the **choose** operation in Line 5 is well-defined as IP-1 and IP-2 hold by our assumption. By Lemma 5.6 the path construction process in Lines 7 through 13 is well-defined as initially IP-2 holds. The execution of Lines 4-18 is therefore well-defined.

To distinguish the state of the variables $\hat{S}, \hat{T}, \hat{f}, \hat{x}$ at Lines 4 and 18 we will use primed variables $\hat{S}', \hat{T}', \hat{f}', \hat{x}'$ to denote the latter state. First note that IP-1 holds by Lemma 5.5: As path P must terminate in either o_S^- or o_r^- (see Lines 5,10), Lemma 5.5 reduces to $f(\delta^+_{E_{\text{ext}}}(v)) - f(\delta^-_{E_{\text{ext}}}(v)) = 0$ for all $v \in V_G$ for $j = |P| - 1$ as neither of the super sinks are included in V_G. The connectivity inequalities IP-2 will also hold with respect to $\hat{S}'$ and $\hat{f}'$ as these are preserved by Lemma 5.6 and $\hat{S}' \subseteq \hat{S}$ holds. Constraint IP-9 holds with respect to $\hat{S}'$ as $\hat{S}' \subseteq \hat{S}$ and the flow along edges in $E^{S^+}_{\text{ext}}$ is never reduced. As similarly flow along edges in $E^{T^+}_{\text{ext}}$ is only reduced for the terminal being connected, Constraint IP-8 could only be violated by a node satisfying $t' \in \hat{T}'$ but $t \notin \hat{T}$. If such a node exists, then it must have been added in Line 16 and as IP-9 has held for $\hat{S}$, constraint IP-3* will hold for $t' \in \hat{S} \cap \hat{T}'$. Analogously, constraint IP-5 is not violated as setting $\hat{x}(s)$ to zero for $s \in \hat{S}$ implies that $s \notin \hat{S}'$ (see Line 16). Constraint IP-4* holds for $\hat{S}'$ as the variable $\hat{x}(s)$ is set to zero whenever the flow along an edge (s, o_S^-) is reduced to zero. Lastly, it is easy to observe that the capacity constraints IP-5, IP-7 cannot be violated as the flow is only reduced. ■

Using Theorem 5.3 we can now prove that Algorithm Decompose terminates.

Theorem 5.7

Algorithm Decompose terminates.

Proof: By iteratively applying Theorem 5.3 the **choose** operations of Algorithm Decompose are well-defined. Note that by construction of the path $|P|$ (see Lines 5,10) flow variables which values are decremented must have been greater or equal to one before the reduction took place. Since the flow $\hat{f} \in \mathbb{Z}_{\geqslant 0}$ is finite and is successively reduced during the process of path construction, the inner loop (see Lines 6-14) must terminate. The outer loop must eventually terminate as well, because each node in $\hat{T}$ (see Line 4) is handled exactly once and as a node $s \in \hat{S}$ may be only moved only once into $\hat{T}$ (see Line 16). ■

Using Theorem 5.3 and 5.7 we can finally prove that Algorithm Decompose indeed constructs a feasible solution for A-CVSAP.

Theorem 5.8

Algorithm Decompose constructs a feasible solution $\hat{\mathcal{T}}_G \in \mathcal{F}_{\text{A-CVSAP}}$ for A-CVSAP given a solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$. Additionally, $C_{\text{CVSAP}}(\hat{\mathcal{T}}_G) \leqslant C_{\text{IP}}(\hat{x}, \hat{f})$ holds.

Proof: To show that for $\hat{\mathcal{T}}_G$ constructed by Algorithm Decompose $\hat{\mathcal{T}}_G \in \mathcal{F}_{\text{M-CVSAP}}$ holds, we need to check CVSAP-1-CVSAP-5 as well as VA-1 and VA-2.We first give short arguments why in fact the conditions CVSAP-1-CVSAP-5 hold:

CVSAP-1	This constraint naturally holds due to Line 2.
CVSAP-2	Algorithm Decompose does not allow for connecting nodes to terminals. Thereby terminals are indeed leaves in $\hat{\mathcal{T}}_G$ and CVSAP-2 holds.
CVSAP-3	Each time another node is connected to the root r the flow along (r, o_r^-) is decremented (see IP-6). As the flow along this edge is bounded by u_r, the degree constraint CVSAP-3 is satisfied by $\hat{\mathcal{T}}_G$
CVSAP-4	An analogue argument as for CVSAP-3 applies.
CVSAP-5	As paths are constructed according to the flow variables $\hat{f}$ that initially respect capacity constraints on edges IP-7, and as $\hat{f}$ is appropriately reduced on used edges, $\hat{\mathcal{T}}_G$ satisfies the edge capacity constraint CVSAP-5.

It remains to prove that $\hat{\mathcal{T}}_G$ satisfies the conditions VA-1 and VA-2 given by in Definition 2.2. As VA-1 follows directly from Line 18, we show consider VA-2 next.

First note that $\hat{T} = \varnothing$ holds when Decompose terminates. We prove that $\hat{S} = \varnothing$ equally holds, thereby showing that each node in $\hat{V}_{\mathcal{T}} \backslash \{r\}$ is connected to another node in $\hat{V}_{\mathcal{T}}$ in Line 15. Assume that $\hat{S} \neq \varnothing$ but $\hat{T} = \varnothing$ holds. We show that this can never be the case using the invariant $s \in \hat{S} \Rightarrow \hat{f}(s, \mathrm{o}_S^-) \geqslant 1$ which directly follows from Theorem 5.3 as IP-4* holds. As this holds for all Steiner nodes, $\hat{f}(\delta^+_{E^{S-}_{\text{ext}}}(\hat{S})) \geqslant |\hat{S}|$ follows. On the other hand, the amount of flow emitted by o^+ equals $|\hat{S}|$ as we assume $\hat{T} = \varnothing$ to hold and by Theorem 5.3 the constraints IP-9 and IP-8 must hold. Due to the flow conservation constraint IP-1, this implies $\hat{f}(r, \mathrm{o}_r^-) \leqslant 0$ which immediately violates Constraint IP-2 by considering the node set $W = V_G$. As this contradicts the statement of Theorem 5.3, we conclude that $\hat{S} = \hat{T} = \varnothing$ must hold when terminating, implying that for all included nodes (see Line 2) an edge was introduced in $\hat{E}_{\mathcal{T}}$ (see Line 18).

As each node (except for the root) has one outgoing edge, it remains to show that $\hat{\mathcal{T}}_G$ does not contain cycles. This follows immediately from the order in which nodes are extracted from $\hat{T}$. This order in fact defines a topological ordering on $\hat{V}_{\mathcal{T}}$ as a cycle containing nodes u and v would imply that u was connected before v and vice versa, that v was connected before u. As this can never be the case, this concludes the proof that $\hat{\mathcal{T}}_G \in \mathcal{F}_{\text{A-CVSAP}}$ holds.

Lastly, $C_{\text{CVSAP}}(\hat{\mathcal{T}}_G) \leqslant C_{\text{IP}}(\hat{x}, \hat{f})$ is valid as costs associated with activating Steiner nodes are incurred in both objectives and Decompose uses only edges already accounted for in $C_{\text{IP}}(\hat{x}, \hat{f})$. In fact $C_{\text{CVSAP}}(\hat{\mathcal{T}}_G) < C_{\text{IP}}(\hat{x}, \hat{f})$ may only be the case if the function `simplify` (see Line 18) truncated a path. ■

To prove that our formulation IP-A-CVSAP indeed computes an optimal solution, we need the following lemma showing that each solution to A-CVSAP can be mapped on a solution of IP-A-CVSAP with equal cost:

Lemma 5.9: Given a network $G = (V_G, E_G, c_E, u_E)$, a request $R_G = (r, S, T, u_r, c_S, u_S)$ and a feasible solution $\hat{\mathcal{T}}_G = (\hat{V}_\mathcal{T}, \hat{E}_\mathcal{T}, r, \hat{\pi})$ to the corresponding A-CVSAP. There exists a solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$ with $C_{\text{CVSAP}}(\hat{\mathcal{T}}_G) = C_{\text{IP}}(\hat{x}, \hat{f})$.

Proof: We define the solution $(\hat{x}, \hat{f}) \in \{0,1\}^S \times \mathbb{Z}_{\geqslant 0}^{E_{\text{ext}}}$ in the following way

- $\hat{x}_s = 1$ iff. $s \in \hat{V}_\mathcal{T}$ for all $s \in S$,
- $\hat{f}_e \triangleq |(\hat{\pi}(\hat{E}_\mathcal{T}))[e]|$ for all $e \in E_G$,
- $\hat{f}_e = 1$ if $v \in \hat{V}_\mathcal{T} \backslash \{r\}$ for all $e = (\text{o}^+, v) \in E_{\text{ext}}^{S^+} \cup E_{\text{ext}}^{T^+}$ and $\hat{f}_e = 0$ otherwise,
- $\hat{f}_e \triangleq \delta^-_{\hat{E}_\mathcal{T}}(s)$ for all $e = (s, \text{o}_S^-) \in E_{\text{ext}}^{S^-}$ and $\hat{f}(r, \text{o}_r^-) \triangleq \delta^-_{\hat{E}_\mathcal{T}}(r)$.

Checking that $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$ and $C_{\text{CVSAP}}(\hat{\mathcal{T}}_G) = C_{\text{IP}}(\hat{x}, \hat{f})$ holds is straightforward. ■

Finally, we can now prove that VirtuCast solves CVSAP to optimality.

Theorem 5.10: ***Algorithm VirtuCast solves A-CVSAP to optimality.***

> *Algorithm VirtuCast, that first computes an optimal solution to IP-A-CVSAP and then applies Decompose, solves A-CVSAP to optimality.*

Proof: We use IP-A-CVSAP to compute an optimal solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$ and afterwards construct the corresponding $\hat{\mathcal{T}}_G \in \mathcal{F}_{\text{A-CVSAP}}$ via Decompose. Assume for the sake of deriving a contradiction that $\hat{\mathcal{T}}_G$ is not optimal and there exists $\tilde{\mathcal{T}} \in \mathcal{F}_{\text{A-CVSAP}}$ with $C_{\text{CVSAP}}(\tilde{\mathcal{T}}) < C_{\text{CVSAP}}(\hat{\mathcal{T}}_G)$. By Lemma 5.9 any solution for A-CVSAP can be mapped on a feasible solution of IP-A-CVSAP of the same objective value. This contradicts the optimality of $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$ and $\hat{\mathcal{T}}_G$ must therefore be optimal. ■

5.3. Runtime Analysis for Decompose

We conclude our discussion of VirtucCast with stating that each **choose** operation in Decompose and checking whether connectivity inequalities IP-2 hold can be implemented using depth-first search. Implementing Decompose in this way and assuming that an optimal solution for IP-A-CVSAP is given and that G does not contain zero-cost cycles, we can bound the runtime from above as follows.

Theorem 5.11

> *Using depth-first search for choosing paths in Algorithm IP-A-CVSAP and for determining whether connectivity inequalities IP-2 are violated, we can bound the runtime by $\mathcal{O}\left(|V_G|^2 \cdot |E_G| \cdot (|V_G| + |E_G|)\right)$, given an optimal solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$ and assuming that graph G does not contain zero-cost cycles.*

Proof: We use depth-first search to separate the connectivity inequalities IP-2 in a canonical manner which we only explain briefly. Given an activated Steiner node $s \in \hat{S}$, we compute the set of all reachable nodes R via depth-first search. If o_r^- is contained in R, then no set of nodes $W \subseteq V_G$ containing s can violate IP-2. On the other hand, if o_r^- is not contained in R, then obviously $W \triangleq R$ violates IP-2. Checking the connectivity inequalities in Line 8 can therefore be performed in time $\mathcal{O}(|\hat{S}| \cdot (|V_{\text{ext}}| + |E_{\text{ext}}|))$. The runtime for choosing a path in Line 10 is clearly dominated by the runtime for checking the connectivity inequalities, and as the previous depth-first search provides a node set W, we do not consider these operations.

The length of any used path P is bounded by $|E_{\text{ext}}|$ as otherwise P would contain a cycle with positive cost. As this cycle can be removed (see function `simplify` in Line 18) yielding a better objective value while remaining feasible, this may never occur by the assumption that our solution is optimal.

Thus, the runtime for the inner loop (Lines 6-14) amounts to $\mathcal{O}(|E_{\text{ext}}| \cdot |\hat{S}| \cdot (|V_{\text{ext}}| + |E_{\text{ext}}|))$ Lastly, the outer loop is performed at most $|\hat{S}| + |\hat{T}|$ many times and the runtime for choosing path P in Line 5 is clearly dominated by the runtime of the inner loop. As $|E_{\text{ext}}| \in \Theta(E_G)$ (assuming E_G to be connected), $|V_{\text{ext}}| \in \Theta(E_G)$ and $|\hat{S}| + |\hat{T}| \in \Theta(V_G)$ holds, the runtime of Decompose is bounded by

$$\mathcal{O}\left(|V_G|^2 \cdot |E_G| \cdot (|V_G| + |E_G|)\right)$$

and our claim follows. ■

As the above theorem only holds for *optimal* solutions and graphs that do not contain zero-cost cycles, we shortly argue that the runtime of Algorithm Decompose is still polynomial in the general case.

Lemma 5.12: *(Polynomial runtime of Algorithm Decompose)* Even if the solution is not optimal and the graph contains zero-cost cycles, the runtime of Algorithm Decompose is still polynomial.

Proof: Without loss of generality, we can assume that $u_E(e) \leqslant |V_G|$ holds for all $e \in E_G$ as each node of V_G will be connected at most to a single other node. As paths of the virtual arborescence are defined to be simple, bounding the edge capacities in this way from above, still allows each node to use each edge on its path once.

In each iteration of the inner loop of Algorithm Decompose, the flow is either reduced in Line 7 or in Line 11. Therefore, and as an edge with flow value 0 will never be used in a path as paths are computed in $G_{\text{ext}}^{\hat{f}}$, the Lines 7 - 13 are at most executed $|V_G| \cdot |E_G|$ times. As the runtime of the inner loop dominates the runtime of the residual algorithm, Algorithm Decompose has a polynomial runtime even if the input solution is not optimal and the graph G contains zero-cost cycles. ■

5.4. Implementation

We have implemented VirtuCast based on IP-A-CVSAP and Decompose. The implementation can be obtained from [RS13a]. Our solver uses SCIP [Ach09] as underlying branch-and-cut

framework with CPLEX [CPL13] as LP solver. Due to space constraints, we will not detail the inner workings of SCIP or general branch-and-bound algorithms but note that the dissertation of Achterberg [Ach07] provides a thorough introduction to these topics.

We therefore constrain our discussion to the customizations made to the SCIP framework. In Section 5.4.1 the implementation of separation procedures for Constraints IP-2 and IP-3* is shortly outlined. Section 5.4.2 details our general adaptions to the employed branching schemes and shortly outlines a custom branching scheme that will be used in the computational evaluation.

5.4.1. Separation

Our solver generally follows the comprehensive work by Koch et al. [KM98] and we assume the reader's familiarity with separation procedures (see e.g. [Sch98]). As the separation techniques used are well-known, we only sketch the most important features.

Instead of using a sophisticated maximal flow algorithm as [KM98] proposes, we implemented the algorithm of Edmonds and Karp (see e.g. [AMO93]). As choosing this simple algorithm only allows for constructing $s-t$ flows, we perform a single maximal flow computation for each $s \in S$ when separating connectivity inequalities IP-2 and analogously perform $|T|$ many maximal flow computations when the valid inequalities of IP-3* are to be separated. To improve performance for executing the maximal flow computations at each node, we use multithreading to speed up the computation.

Furthermore, we have implemented techniques that *can* (empirically) improve the *quality* of found violated inequalities for IP-2 or IP-3*. Following [KM98] we implemented the techniques of *creep-flow* and *nested cuts*. Creep-flow places a minimal amount of flow on edges even when they are not used at all, such that edges without flow on it will not automatically be placed in cuts. Nested cuts on the other hand is a technique to obtain multiple cuts within a *single* separation round. If a valid cut was found, the flow on edges of the cut is set to 1 and a new cut is searched for. As the previous maximum flow computationc an be used to find the new cuts, i.e. the algorithm can be warm started, and the computational cost for finding further cuts is comparatively small.

We opted not to implement *back cuts*, as in our formulation of IP-2 violated node sets with respect to a given Steiner site $s \in S$ would probably be violated for other Steiner sites too. Adding back cuts for each of the violated node sets with respect to many $s \in S$ would in turn probably lead to many redundant constraints.

In Section 11 the impact of creep-flow, nested cuts and the separation of terminal connectivity inequalities is investigated.

5.4.2. Branching

Branching procedures decide in which way a problem shall be splitted into (disjoint) subproblems by creating several nodes in the branch-and-bound tree and changing some variables' upper or lower bounds. If branching is applied to a single binary variable, the variable will be set to zero in one child and to one in the other.

As a general decision that applies to all branching schemes, we assign higher priorities to Steiner site opening variables $x \in \{0,1\}$ than to flow variables $f_e \in \mathbb{Z}_{\geqslant 0}$. Therefore, all branching schemes will first apply branching rules on Steiner site opening variables. While we only validated this decision as part of the first (undocumented) experimentations, the reasoning behind this is convincing:

1. Costs associated with Steiner sites are generally much higher than costs of edges. Activating a Steiner site is therefore more likely to yield an objective value that is larger than the primal bound, such that this node can be cut off.

2. On the other hand, disabling a Steiner site $s \in S$ by fixing $x_s = 0$ is more likely to yield an infeasible solution, as formerly absorbed flow needs to be rerouted.

3. Generally, only after having fixed all Steiner site opening variables, the amount of flow emitted from the super source o^+ into the network is fixed.

We have also implemented an own branching scheme, that is introduced in Definition 5.13. The underlying idea is to first branch on Steiner sites that maximally contribute to the objective and then to branch on edge variables that maximize the distance to the nearest lower integer.

Definition 5.13: GREEDY BRANCHING RULE
Given a fractional solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{LP}}$, the greedy branching rule operates as follows.

1. If a fractional Steiner site variable exists, branch on $s \in S$ with $\hat{x}_s \in (0,1)$ maximizing $c_S(s) \cdot \hat{x}_s$.

2. If no fractional Steiner site variable exists, branch on the edge $e \in E_{\text{ext}}$ maximizing $\hat{f}_e - \lfloor \hat{f}_e \rfloor$, by creating two childs with additional constraints $\hat{f}_e \leqslant \lfloor \hat{f}_e \rfloor$ and $\hat{f}_e \geqslant \lceil \hat{f}_e \rceil$ respectively.

Part III.

Heuristics for CVSAP

6. Overview and Common Algorithms

In Part II, two exact algorithms based on Integer Programming (IP) have been proposed to solve CVSAP. Employing branch-and-bound (or branch-and-cut) techniques (see [Sch98] for an introduction), the proposed IP formulations can be solved by solvers as CPLEX [CPL13] or SCIP [Ach09] to optimality in non-polynomial time. To obtain feasible solutions in polynomial time, IP solvers employ general and problem-specific heuristics [Ach09]. While finding feasible solutions *within reasonable time* with an guarantee on its quality is the top-most objective, finding good solutions also allow to speed-up the solution process, as will be discussed below.

Determining an optimal solution (or proving that none exists) is guaranteed by systematically exploring the whole solution space via *branching* procedures that subdivide the solution space. However, while branching reduces the solution space, the number of (reduced) problems spawned increases exponentially. To efficiently solve IPs, bounding procedures are employed. Given a subproblem, a dual bound is computed that bounds the best possible solution value by considering a relaxation of the problem that can be solved in polynomial time. In our case the relaxation employed is the linear one that is obtained by relaxing the integrality of variables. Given a feasible (primal) solution to the overall problem, the primal bound is defined by the (best) solution's objective. Clearly, if the dual bound of a subproblem is worse than the primal bound, then the corresponding solution space does not need to be considered and can be *cut off*, thereby potentially reducing the search space significantly.

As the heuristics built into the SCIP framework often fail to produce high-quality solutions (see Section 14.3) we propose several heuristics to (try to) obtain feasible solutions for CVSAP. With one exception, all proposed heuristics rely on the formulation IP-A-CVSAP and the ability to solve it quickly (see Section 8). We furthermore initiate the study of combinatorial heuristics by introducing the IP-independent heuristic GreedySelect which is presented in Section 7.

Before presenting the heuristics in Sections 7 and 8, first notations related to standard algorithms employed are defined in Section 6.1. In Section 6.2 a polynomial local search procedure will be introduced to improve the objective value of a feasible virtual arborescence by pruning active Steiner nodes.

6.1. Employed Known Algorithms and Definitions

The heuristics presented in Sections 8 and 7 will make use of both shortest paths and minimum-cost flow algorithms. As these algorithms are well-known [KV12] we only define their interface, i.e. their inputs and outputs. At the end of this section we will also define the notion of partial virtual arborescences.

Definition 6.1: FUNCTION `ShortestPath`$(G, c, s, \mathcal{D})$

Input: Directed network $G = (V_G, E_G, c_E)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$, a start node $s \in V_G$ and a set of destinations $\varnothing \neq \mathcal{D} \subseteq V_G$.

Output: Directed path $P = \langle P_1, P_2, ..., P_n \rangle$, such that $P_1 = s$ and $P_n \in \mathcal{D}$ holds and its cost $c_E(P) \triangleq \sum_{i=1}^{n-1} c_E(P_i, P_{i+1})$ is minimal or `null` if no such path exists.

Remark 6.2 *(Runtime of `ShortestPath` [KV12])***.** A shortest path can be computed in time $\mathcal{O}(|E_G| + |V_G| \log |V_G|)$ by using Dijkstra's algorithm.

As a simple extension, of `ShortestPath`, we will use `MinAllShortestPath` to compute the shortest path between a pair of sets of nodes.

Definition 6.3: FUNCTION `MinAllShortestPath`$(G, c, \mathcal{S}, \mathcal{D})$

Input: Directed network $G = (V_G, E_G, c_E)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$, a set of start nodes $\mathcal{S} \subseteq V_G$ and a set of destinations $\varnothing \neq \mathcal{D} \subseteq V_G$.

Output: Directed path $P = \langle P_1, P_2, ..., P_n \rangle$, such that $P_1 \in \mathcal{S}$ and $P_n \in \mathcal{D}$ holds and its cost $c_E(P) \triangleq \sum_{i=1}^{n-1} c_E(P_i, P_{i+1})$ is minimal or `null` if no such path exists.

Remark 6.4 *(Runtime of `MinAllShortestPath`)***.** The problem of finding the shortest path between two sets of nodes can be reduced to a single shortest path computation: introduce two additional nodes of whom the first connects to all nodes in $\mathcal{S}$ and all nodes of $\mathcal{D}$ connect to the second one and compute a shortest path from the first to the second node.

Definition 6.5: FUNCTION `MinCostFlow`$(G, c_E, u_E, v, s, \mathcal{D})$

Input: Directed network $G = (V_G, E_G)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$ and integral capacities $u_E : E_G \to \mathbb{N}$, a start node $s \in V_G$, a set of destinations $\varnothing \neq \mathcal{D} \subseteq V_G$ and the flow value $v \in \mathbb{N}_{>0}$ that shall be achieved.

Output: Set of paths $\Gamma = \{(P_i, f_i)\} \in \mathcal{P}_G \times \mathbb{N}_{>0}$ with a value f_i specifying the amount of flow carried by it, such that

a) P_i starts in t and ends at some node of $\mathcal{D}$,

b) the decomposition does not violate capacities, i.e. $\sum_{e \in P_i} f_i \leqslant u_E(e)$ holds for all edges $e \in E_G$, and

c) the decomposition has value v, i.e. $\sum f_i = v$ holds and

d) the cost of the decomposition $c_E(\Gamma) \triangleq \sum_{(P_i, f_i) \in \Gamma} f_i \cdot c_E(P_i)$ is minimal.

If no flow (decomposition) Γ of value v exists, then $\varnothing$ is returned.

The following remark establishes the polynomial runtime of `MinCostFlow`.

Remark 6.6 *(Runtime of `MinCostFlow`, [KV12])***.** By using the successive shortest paths algorithm for computing a mininmal cost flow and decomposing it afterwards using e.g. breadth-first search, $\mathcal{O}(|V_G| \cdot |E_G| + v(|E_G| + |V_G| \log |V_G|))$ is an upper bound on the runtime of `MinCostFlow`.

We will also make use of the following algorithm to solve a minimum-cost assignment problem.

Definition 6.7: FUNCTION `MinCostAssignment`$(G, c_E, u_E, u_V, \mathcal{S}, \mathcal{D})$

Input: Directed network $G = (V_G, E_G)$ with edge costs $c_E : E_G \to \mathbb{R}_{\geqslant 0}$ and integral capacities on edges $u_E : E_G \to \mathbb{N}$ and end nodes $u_{\mathcal{D}} : \mathcal{D} \to \mathbb{N}$, a set of nodes that shall be connected $\mathcal{S} \subset V_G$ and a set of destinations $\varnothing \neq D \subseteq V_G$.

Output: Set of paths $\Gamma = \{P^s | s \in \mathcal{S}\} \in \mathcal{P}_G$, such that

a) P_s starts in s and ends at some node of $\lceil$,

b) paths do not violate edge capacities, i.e. $|\{P^s | P^s \in \Gamma, e \in P_s\}| \leqslant u_E(e)$ holds for all edges $e \in E_G$, and

c) paths do not violate node capacities, i.e. $|\{P^s | P^s \in \Gamma, P^s \text{ ends in } d\}| \leqslant u_{\mathcal{D}}(d)$ for all destinations $d \in \mathcal{D}$,

d) the cost of the decomposition $c_E(\Gamma) \triangleq \sum_{P^s \in \Gamma} c_E(P^s)$ is minimal.

If no such assignment exists $\varnothing$ is returned.

By a simple construction `MinCostAssignment` can be reduced to solving a minimum-cost flow problem by introducing two additional nodes and connecting them to $\mathcal{S}$ and $\mathcal{D}$ respectively (see Remark 6.4). By defining the capacity of new edges towards $\mathcal{S}$ to be one and by defining the capacity of new edges leaving $\mathcal{D}$ to be equal to the nodes capacity, the problem can be easily reduced onto a minimum-cost flow computation of value $|\mathcal{S}|$. We therefore obtain the following remark.

Remark 6.8 *(Runtime of `MinCostAssignment`)***.** As $\mathcal{S} \subset V_G$ holds the runtime of Algorithm `MinCostAssignment` is bounded by $\mathcal{O}(|V_G| \cdot |E_G| + |V_G|^2 \log |V_G|)$ via the above described reduction to a minimum-cost flow problem.

6.2. Local Search Procedure PruneSteinerNodes ★

In this section the Algorithm PruneSteinerNodes (see Algorithm 6.1) is introduced. Algorithm PruneSteinerNodes iteratively tries to remove the active Steiner node which has the worst ratio of installation costs divided by the number of incoming connections. Having removed the Steiner node and its connection, some nodes, including Steiner nodes, are unconnected. To reconnect these nodes we iteratively compute shortest paths towards active Steiner nodes (or the root) with remaining node capacity. Importantly, when reconnecting an active Steiner node, only shortest paths are considered which do not yield a cycle in the virtual arborescence.

In the following, we explain the algorithm's implementation. Initially, the activated Steiner nodes are put in the set O (see Line 1). According to the ratio of cost for installing it divided by the number of nodes connected to it, the Steiner node maximizing this ratio is selected (see Line 3). Together with all its incoming and outgoing edges, it is removed from the solution, yielding a (temporarily infeasible) solution $(\hat{V}'_{\mathcal{T}}, \hat{E}'_{\mathcal{T}}, r, \hat{\pi}')$ and the remaining capacity $u' : E_{\text{ext}} \to \mathbb{N}$ (see Lines 9 to 12). By removing paths and disabling the Steiner node, the objective value of the virtual arborescence is decreased, giving an budget b for reconnecting the disconnected nodes (see Line 8). Reconnecting the disconnected nodes is done using shortest paths under the constraint that no cycles may be introduced to the solution (see Lines 13 to 20). This constraint can be easily realised by first computing all active Steiner nodes that are in the same connected component as the (Steiner) node that needs to be connected and removing these edges when computing the shortest path.

If a node cannot be connected or using the shortest path would exceed the budget (see Line 15), the algorithm selects another activated Steiner node, if possible (see Line 16). If however all nodes could be reconnected and the budget was not exceeded, then a cheaper feasible virtual arborescence has been found and the process is restarted with all opened aggregation nodes (see Lines 21 and 22). We conclude by shortly showing the polynomial runtime of Algorithm PruneSteinerNodes.

Theorem 6.9: ***Polynomial runtime of Algorithm PruneSteinerNodes.***

The runtime of Algorithm PruneSteinerNodes is bounded by $\mathcal{O}(|V_G|^2 \cdot |E_G| + |V_G|^3 \log |V_G|))$.

Proof: First note that the set of opened Steiner nodes O is only recomputed in Line 22, if the algorithm did not abort when reconncting nodes. Hence one Steiner node was removed. Thus, the body of the while loop (Lines 3-22) is executed at most $|O|^2 \leqslant |V_G|^2$ many times.

We now show that between two successul removals of Steiner nodes at most V_G many shortest paths computations were performed. This holds, as each node in $v \in \hat{V}_{\mathcal{T}}$ is connected to at most one node $v' \in \hat{V}_{\mathcal{T}}$. Hence, a shortest paths computation from node v is only triggered if node v' was removed (temporarily). Therefore, between two successful Steiner node removals node v' will only be considered once and maximally $|O| \leqslant |VG|$ many shortest paths computations are performed between successful Steiner node removals. As only $|O| \leqslant |V_G|$ many nodes can be removed, the overall runtime of Lines 13-20 can be bounded by $\mathcal{O}(|V_G|^2 \cdot (|E_G| + |V_G| \log |V_G|)$. As this runtime dominates the runtime of the possibly $|V_G|^2$ many executions of Lines 3-12 and the $|VG|$ many executions of Lines 21,22, the theorem

follows. ■

Algorithm 6.1: PruneSteinerNodes

Input : Network $G = (V_G, E_G, c_E, u_E)$, Request $R_G = (r, S, T, u_r, c_S, u_S)$,
Solution $\hat{\mathcal{T}}_G \in \mathcal{F}_{\text{A-CVSAP}}$ for A-CVSAP

Output: Feasible Virtual Arborescence $\hat{\mathcal{T}}'_G \in \mathcal{F}_{\text{A-CVSAP}}$ with $C_{\text{CVSAP}}(\hat{\mathcal{T}}'_G) \leqslant C_{\text{CVSAP}}(\hat{\mathcal{T}}_G)$

set $O \triangleq S \cap \hat{V}_{\mathcal{T}}$
while $O \neq \varnothing$ **do**
  choose $s \in O$ **maximizing** $c_S(s)/|\delta^-_{\hat{E}_{\mathcal{T}}}(s)|$
  set $O \leftarrow O - s$
  set $U \triangleq \{t | (t,s) \in \delta^-_{\hat{E}_{\mathcal{T}}}(s)\}$
  set $R \triangleq \delta^+_{\hat{E}_{\mathcal{T}}}(s) \cup \delta^-_{\hat{E}_{\mathcal{T}}}(s)$
  set $\mathcal{P}_R \triangleq \{\hat{\pi}(e) | e \in R\}$
  set $b \triangleq c_S(s) + \sum_{P \in \mathcal{P}_R} c_E(P)$
  set $\hat{V}'_{\mathcal{T}} \triangleq \hat{V}_{\mathcal{T}} \backslash (U \cup \{s\})$
  set $\hat{E}'_{\mathcal{T}} \triangleq \hat{E}_{\mathcal{T}} \backslash (\delta^-_{\hat{E}_{\mathcal{T}}}(s) \cup \delta^+_{\hat{E}_{\mathcal{T}}}(s))$ **and** $\hat{\pi}' : \hat{E}'_{\mathcal{T}} \to \mathcal{P}_G$
    such that $\hat{\pi}'(e) = \hat{\pi}(e)$ **for all** $e \in \hat{E}'_{\mathcal{T}}$
  set $u'(e) \triangleq \begin{cases} u_E(e) - |\hat{\pi}(\hat{E}'_{\mathcal{T}})[e]| & \text{, if } e \in E_G \\ u_r(r) - |\delta^-_{\hat{E}'_{\mathcal{T}}}(r)| & \text{, if } e = (r, \mathrm{o}_r^-) \\ u_S(s) - |\delta^-_{\hat{E}'_{\mathcal{T}}}(s')| & \text{, if } e = (s', \mathrm{o}_S^-) \in E^{S^-}_{\text{ext}} \\ 1 & \text{, else} \end{cases}$ **for all** $e \in E_{\text{ext}}$
  set $u'(s, \mathrm{o}_S^-) \leftarrow 0$
  for $e = (t,s) \in \delta^-_{\hat{E}_{\mathcal{T}}}(s)$ **do**
    choose $P \triangleq \texttt{ShortestPath}\left(G^{u'}_{\text{ext}}, c_E, t, \{\mathrm{o}_S^-, \mathrm{o}_r^-\}\right)$
      such that $(\hat{V}'_{\mathcal{T}} + t, \hat{E}'_{\mathcal{T}} + (t, P_{|P|-1}))$ is acyclic
    if $P = \varnothing \vee b - c_E(P) \leqslant 0$ **then**
      goto 2
    set $b \leftarrow b - c_E(P)$
    set $\hat{V}'_{\mathcal{T}} \leftarrow \hat{V}'_{\mathcal{T}} + t$
    set $\hat{E}_{\mathcal{T}} \leftarrow \hat{E}_{\mathcal{T}} + (t, P_{|P|-1})$ **and** $\hat{\pi}(t, P_{|P|-1}) \triangleq \langle P_1, \ldots, P_{|P|-1} \rangle$
    set $u'(e) \leftarrow u'(e) - 1$ **for all** $e \in P$
  set $\hat{\mathcal{T}}_G \leftarrow$ Virtual Arborescence $(\hat{V}'_{\mathcal{T}}, \hat{E}'_{\mathcal{T}}, r, \hat{\pi}')$
  set $O \leftarrow S \cap \hat{V}_{\mathcal{T}}$
return $\hat{\mathcal{T}}_G$

7. Combinatorial Heuristic GreedySelect

In this section we present a purely combinatorial heuristic for CVSAP that does not rely on linear programming methods to find solutions. Following an greedy approach Algorithm GreedySelect iteratively connects one or multiple nodes in a single iteration such that the connection cost *per connected node* is minimized. We allow for the following types of connections.

1. An unconnected node (either terminal or active Steiner node) may be connected to an active Steiner node that is already connected to the root or to the root itself, such that only one node is connected.

2. A single inactive Steiner node can be activated, such that multiple unconnected nodes (either terminals or active Steiner nodes) can be connected to it. Note that the Steiner node which is activated, will not be connected.

We associate the following costs with each of the above operations.

1. If only a single node is connected, the cost of this operation is simply the sum of the edge costs of the path used for connecting the node.

2. If an inactive Steiner node $\bar{s}$ is activated and a set of unconnected nodes T' is connected to it, the following cost model is applied. For each of the nodes T' the costs of their respective paths to connect to $\bar{s}$ is added to the installation costs of $\bar{s}$, yielding the costs $c_{T',\bar{s}}$. Note that $c_{T',\bar{s}}$ equals the costs that will be introduced into the solution. As node $\bar{s}$ will need to be connected later on to the connected component R that raches r, we denote by $c_{\bar{s},R}$ the shortest path cost to connect $\bar{s}$ to R. Lastly, we denote by $c_{T',R}$ the sum of shortest path costs to connect each of the nodes in T' to R directly. As this cost will be avoied, we subtract it, yielding the following cost per connected node

$$\left(c_{T',\bar{s}} + c_{\bar{s},R} - c_{T',R}\right) / |T'| \,.$$

In the following we first give a synopsis of Algorithm GreedySelect in Section 7.1 and then prove the polynomial runtime in Section 7.2.

7.1. Synopsis of Algorithm GreedySelect

In Line 1 the sets $\bar{S}, \bar{T}, \hat{S}, \hat{T}$ are initialized. The set $\bar{S}$ will contain inactive and therefore unconnected Steiner nodes. The set $\bar{T}$ stores all unconnected Terminals *and* all activated,

Algorithm 7.1: GreedySelect

Input : Network $G = (V_G, E_G, c_E, u_E)$, Request $R_G = (r, S, T, u_r, c_S, u_S)$
Output: A Feasible Virtual Arborescence $\hat{\mathcal{T}}_G$ or `null`

1 **set** $\bar{S} \triangleq S$, $\bar{T} = T$ **and** $\hat{T} \triangleq \varnothing$, $\hat{S} \triangleq \varnothing$
2 **set** $\hat{\mathcal{T}}_G \triangleq (\hat{V}_\mathcal{T}, \hat{E}_\mathcal{T}, r, \hat{\pi})$, **where** $\hat{V}_\mathcal{T} \triangleq \{r\}$, $\hat{E}_\mathcal{T} \triangleq \varnothing$ **and** $\hat{\pi} : \hat{E}_\mathcal{T} \to \mathcal{P}_G$
3 **set** $u(e) \triangleq u_E(e)$ **for all** $e \in E_G$
5 **while** $|\bar{T}| > 0$ **do**
6 **compute** $R \leftarrow \{r' \in \{r\} \cup \hat{S} | r'$ reaches r in $\hat{\mathcal{T}}_G, \delta^-_{\hat{\mathcal{T}}_G}(r') < u_{r,S}(r')\}$
7 **compute** $P_{v,R} = \texttt{ShortestPath}(G^u, c_E, v, R)$ for $v \in \bar{S} \cup \bar{T}$
8 **if** $P_{\bar{t},R} = \texttt{null}$ **for some** $\bar{t} \in \bar{T}$ **then**
9 **return** `null`
10 **end**
11 **set** $\hat{t} \leftarrow \bar{t} \in \bar{T}$, **such that** $c_E(P_{\bar{t},R}) \leqslant c_E(P_{\bar{t}',R})$ for all $\bar{t}' \in \bar{T}$
12 **set** $u^{\bar{s}}(e) \triangleq u(e) - |P_{\bar{s}}[e]|$ **for all** $\bar{s} \in \bar{S}, e \in E_G$
13 **compute** $\mathcal{P}_{\bar{s}} \triangleq (\bar{s} \in \bar{S}, T' \subseteq \bar{T}, \mathcal{P}_{T'} = \{P_{t,\bar{s}} | t \in T'\})$,

such that $P_{t,\bar{s}}$ connects t to $\bar{s}$,
$u^{\bar{s}}(e) - |\mathcal{P}_{T'}[e]| \geqslant 0$ for all $e \in E_G$,
$2 \leqslant |T'| \leqslant u_{r,S}(\bar{s})$

minimizing $c_{\bar{s},T'} \triangleq \left(\sum_{t \in T'} (c_E(P_{t,\bar{s}}) - c_E(P_{t,R})) + c_E(P_{\bar{s},R}) + c_S(\bar{s}) \right) / |T'|$

14 **if** $\mathcal{P}_{\bar{s}} \neq \texttt{null}$ ***and*** $c_{\bar{s},T'} < c_E(P_{\hat{t},R})$ **then**
15 **set** $\bar{T} \leftarrow (\bar{T} + \bar{s}) \backslash T'$ **and** $\bar{S} \leftarrow \bar{S} - \bar{s}$
16 **set** $\hat{V}_\mathcal{T} \leftarrow \hat{V}_\mathcal{T} \cup (T' + \bar{s})$
17 **set** $\hat{E}_\mathcal{T} \leftarrow \hat{E}_\mathcal{T} + \{(t, \bar{s}) | t \in T'\}$ **and** $\hat{\pi}(t, \bar{s}) \triangleq P_{t,\bar{s}}$ **for all** $t \in T'$
18 **else**
19 **set** $\bar{T} \leftarrow \bar{T} - \hat{t}$
20 **set** $\hat{V}_\mathcal{T} \leftarrow \hat{V}_\mathcal{T} + \hat{t}$, $\hat{E}_\mathcal{T} \leftarrow \hat{E}_\mathcal{T} + (t, P_{|P_{\hat{t}}|})$ **and** $\hat{\pi}(t, P_{|P_{\hat{t}}|}) \triangleq P_{\hat{t}}$ **for all** $t \in T'$
21 **end**
22 **end**
23 **return** `PruneSteinerNodes` $(\hat{\mathcal{T}}_G)$

but unconnected, Steiner nodes. The sets $\hat{S}$ store the activated and the connected terminals respectively. As initially no node is connected, capacities $u : E_G \to \mathbb{N}$ are intialized to equal the original capacities in Line 3. The while loop from Line 5 to Line 22 then tries to connect nodes as long as the set of unconnected nodes $\bar{T}$ is not empty.

First the set of nodes $R \subseteq \hat{V}_\mathcal{T}$ is computed, such that R contains only the activated Steiner nodes (or the root r) that are already connected to r. Note that nodes whose (node) capacities are fully used, are not included in R (see Line 6). Then, a shortest path $P_{v,R}$ is computed for

each inactive Steiner node and each unconnected terminal towards R. The algorithm aborts in Lines 8-10, if there exists an unconnected terminal that does not have a path towards R. This abort is necessary, as Algorithm GreedySelect does not perform any path reconfigurations and can therefore not recover once a terminal has been disconnected from the root.

In Line 11 the unconnected node $\hat{t} \in \bar{T}$ is selected that has the minimal shortest path towards R. Note that $\hat{t}$ may either be a terminal or an active Steiner node. In Line 12 edge capacities $u^{\bar{s}}$ are computed for each $\bar{s} \in \bar{S}$, such that the capacity on an edge e is decremented by one if and only if its shortest path $P_{\bar{s},R}$ used edge e.

In the Line 13 the cheapest Steiner node $\bar{s} \in \bar{S}$ to activate is computed together with a set of unconnected nodes T' and a path $P_{t,\bar{s}}$ for each $t \in T'$ to connect to $\bar{s}$. We require that the set of paths $\mathcal{P}_{T'}$ does not violate the edge capacities $u^{\bar{s}}$, such that node $\bar{s}$ can still be connected to R if all paths in $\mathcal{P}_{T'}$ are established. As the number of subsets $T' \subset \bar{T}$ may be exponential, we will show in the next section how this computation can be implemented in polynomial time.

If a minimum cost tuple $\mathcal{P}_{\bar{s}}$ was found such that its cost is below the cost of shortest path $P_{\hat{t},R}$, then all nodes T' are connected to $\bar{s}$ in Lines 15-17 and $\bar{s}$ is put into the set of unconnected nodes $\bar{T}$. Otherwise the shortest path $P_{\hat{t},R}$ is used to connect $\hat{t} \in \bar{T}$ in Lines 19 and 20.

Lastly, if the algorithm does not abort in Line 9, in Line 23 the constructed virtual arborescence $\hat{\mathcal{T}}_G$ is returned after having pruned the solution.

7.2. Runtime of Algorithm GreedySelect

It is easy to check that in Algorithm GreedySelect the number of unconnected terminals $\bar{T}$ is at least reduced by one in each iteration: either an unconnected node is directly connected or multiple unconnected nodes are connected to a single newly unconnected node $\bar{s}$.

To bound the runtime and show that it is indeed polynomial, we have to prove that the computation of $\mathcal{P}_{\bar{s}}$ can be implemented in polynomial time. We show this in the following lemma.

Lemma 7.1: *(Runtime of Line 13)* The computation of $\mathcal{P}_{\bar{s}}$ can be implemented in time $\mathcal{O}(|V_G| \cdot |E_G| + |V_G|^2 \log |V_G|))$.

Proof: Note first that the costs $c_E(P_{\bar{s},R})$ and $c_S(\bar{s})$ are independent of the set of connected terminals T'. Assume for now that the number of terminals $n = |T'|$ to connect is fixed, and we only consider a single inactive Steiner node $\bar{s} \in \bar{S}$. We show that this subproblem can be solved by computing a minimum-cost flow using a similar construction to the one used for reducing `MinCostAssignment` to `MinCostFlow` in Section 6.1: we introduce a single super source that is connected to the set of nodes $\bar{t} \in \bar{T}$ with unit capacities *and costs* $-c_E(P_{\bar{t},R})$ and compute a minimum-cost flow of value n from this super source towards the fixed inactive Steiner node s. To see that a minimum-cost flow algorithm can be employed on this network, we have to prove that edge costs are conservative [KV12], i.e. that the extended network does not contain directed cycles of cost less than 0. As edges with negative costs only leave the super source, which has no incoming edges, this is immediate. Therefore, the corresponding minimum-cost flow problem can be solved by using standard successive

shortest paths algorithms [KV12]. The advantage of employing a successive shortest paths algorithm lies in the fact that computing a minimum-cost flow of value $u_{r,S}\bar{s}$ yields solutions for $|T'| = 2, 3, \ldots, u_{r,S}\bar{s}$ on-the-fly. Therefore, by performing a single successive shortest path minimum-cost flow computation for each inactive Steiner node, the optimal set can be found. The runtime of the computation of $\mathcal{P}_{\bar{s}}$ can therefore (by Remark 6.6) be bounded by $\mathcal{O}(|V_G| \cdot |E_G| + |V_G|^2 \log |V_G|))$. ■

By the above lemma, it is easy to prove the following runtime.

Theorem 7.2: ***Runtime of Algorithm GreedySelect***

Algorithm GreedySelect can be implemented to run in $\mathcal{O}(|V_G|^2 \cdot |E_G| + |V_G|^3 \log |V_G|)$.

Proof: Note that the runtime of all other operations within the while loop is clearly dominated by the runtime of Line 13. As already observed, the number of unconnected terminals $\bar{T}$ is reduced in each iteration by at least one. Hence, we obtain the runtime bound of $\mathcal{O}(|V_G|^2 \cdot |E_G| + |V_G|^3 \log |V_G|)$. As this runtime bound equals the runtime bound of executing Algorithm PruneSteinerNodes in Line 23 (see Theorem 6.9), the theorem is proven. ■

8. LP-Based Heuristics

In this section three different types of heuristics are presented for CVSAP that all rely on the linear relaxation of IP Formulation IP-A-CVSAP and the ability to solve it rather quickly (at least e.g. in comparison to Formulation A-CVSAP-MCF). The first algorithm presented will try to construct a solution given a single LP relaxation while the remaining ones will actively resolve linear relaxations to guide the construction of solutions. There are two main reasons for studying IP-based heuristics in-depth.

1. Even though SCIP implements more than two dozen general IP-based heuristics, our computational evaluation has shown that these heuristics may fail to produce even a single solution for larger instances (see Section 14.3). We believe that this is mainly due to the fact that SCIP does not perfom separation procedures for Constraints IP-2 and IP-3*. This necessitates the development of heuristics to first and foremost generate feasible solutions at all, but also to speed up the execution of the branch-and-*bound* algorithm on smaller instances. As linear relaxations are computed during the branch-and-bound search anyway, they can be used to derive information on e.g. which node connects (fractionally) to which node. The heuristic FlowDecoRound which is presented in Section 8.1 e.g. uses the flow values to derive probabilities for connecting nodes. The heuristics presented in Section 8.6 will utilize the fractional Steiner opening variables to derive probabilities for activating Steiner nodes.

2. Based on Algorithm Decompose a direct connection between feasible solutions of formulation IP-A-CVSAP and the underlying CVSAP instance (cf. Theorem 5.8) has been established. Finding feasible solutions to IP-A-CVSAP seems conceptually easier, since instead of paths only aggregated flow values need to be computed. This is important, as combinatorial algorithms (that iteratively connect nodes as in Section 7) irrevocably reduce the links' capacities during their execution and *cannot* reconfigure paths (to an unlimited extent). Especially the diving heuristics presented in Section 8.5 will make use of the fact that linear relaxations allow for potentially *global* path reconfigurations, by simply not considering an explicit path model.

8.1. Heuristic FlowDecoRound ★

Our primal heuristic FlowDecoRound (see Algorithm 8.1) uses the linear relaxation $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{LP}}$ at the current node in the branch-and-bound tree as input. In the first phase (see Lines 4 to 15) for each terminal a flow decomposition is computed based on the flow values $\hat{f}$ of the current LP solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{LP}}$ such that the path may either terminate in o_S^- or o_r^-. As we only have defined `MinCostFlow` on integral flows, we only compute an approximate flow

decomposition by scaling flows to integers internally. The flow decomposition returns a set of paths paired with an amount of flow carried by them. After discarding paths for which no capacity is left and paths that would lead to a cycle in the solution, one of the remaining paths is chosen uniformly at random according to the flow amount carried by it. If the path leads to an (inactive) Steiner site, then the aggregation node is opened and becomes itself a terminal to be connected during the first phase. If none of the paths returned by the flow decomposition is feasible, the terminal is not connected.

In the second phase (see Lines 18 to 23) the terminals (including Steiner nodes) that are still disconnected are connected using shortest paths under the restriction that these paths may not yield a cycle in the solution.

Lastly, if all terminals (including Steiner nodes) have been connected in the second phase, a feasible solution was constructed. Since in the first phase any Steiner node is activated if a path to it was selected, we call Algorithm PruneSteinerNodes to potentially reduce the objective value. As discussed in Section 12, pruning Steiner nodes can indeed improve the quality of the found solution significantly.

We lastly consider the runtime of Algorithm FlowDecoRound.

Theorem 8.1: ***Runtime of Algorithm FlowDecoRound***

The runtime of Algorithm FlowDecoRound is bound by $\mathcal{O}(|V_G|^2 \cdot |E_G| + |V_G|^3 \log |V_G|)$.

Proof: We note that in the first as well as in the second phase maximally V_G many nodes are considered. Within the first phase, the runtime is dominated by the call of `MinCostFlow`. As we use an integral flow algorithm, but scale the flow beforehand by a constant, the scaling constant increases the runtime only upto a constant. The runtime of the first phase is therefore bounded by $\mathcal{O}(|V_G|^2 \cdot |E_G| + |V_G| \log |V_G|)$ (cf. Remark 6.6).

In the second phase, the runtime is dominated by performing the shortest paths computation in Line 19 and is therefore bounded by $\mathcal{O}(|V_G| \cdot |E_G| + |V_G|^2 \log |V_G|)$ (cf. Remark 6.2). Lastly, the runtime spent in Algorithm PruneSteinerNodes is bound by $\mathcal{O}(|V_G|^2 \cdot (|E_G| + |V_G| \log |V_G|))$ (see Theorem 6.9). Therefore, the overall runtime of Algorithm FlowDecoRound is dominated by the runtime for pruning active Steiner nodes and the theorem follows. ■

Algorithm 8.1: FlowDecoRound

Input : Network $G = (V_G, E_G, c_E, u_E)$, Request $R_G = (r, S, T, u_r, c_S, u_S)$, LP relaxation solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{LP}}$ to IP-A-CVSAP

Output: A Feasible Virtual Arborescence $\hat{\mathcal{T}}_G$ or `null`

set $\hat{S} \triangleq \varnothing$ **and** $\hat{T} \triangleq \varnothing$ **and** $U = T$

set $\hat{V}_{\mathcal{T}} \triangleq \{r\},\ \hat{E}_{\mathcal{T}} \triangleq \varnothing$ **and** $\hat{\pi} : \hat{E}_{\mathcal{T}} \to \mathcal{P}_G$

set $u(e) \triangleq \begin{cases} u_E(e) & \text{, if } e \in E_G \\ u_r(r) & \text{, if } e = (r, \mathrm{o}_r^-) \\ u_S(s) & \text{, if } e = (s, \mathrm{o}_S^-) \in E_{\text{ext}}^{S^-} \\ 1 & \text{, else} \end{cases}$ **for all** $e \in E_{\text{ext}}$

while $U \neq \varnothing$ **do**

  choose $t \in U$ uniformly at random **and set** $U \leftarrow U - t$

  set $\Gamma_t \triangleq \texttt{MinCostFlow}\left(G_{\text{ext}}, \hat{f}, \hat{f}(\mathrm{o}^+, t), t, \{\mathrm{o}_S^-, \mathrm{o}_r^-\}\right)$

  set $\hat{f} \leftarrow \hat{f} - \sum_{(P,f) \in \Gamma_t, e \in P} f$

  set $\Gamma_t \leftarrow \Gamma_t \backslash \{(P, f) \in \Gamma_t | \exists e \in P . u(e) = 0\}$

  set $\Gamma_t \leftarrow \Gamma_t \backslash \{(P, f) \in \Gamma_t | (\hat{V}_{\mathcal{T}} + t, \hat{E}_{\mathcal{T}} + (t, P_{|P|-1})) \text{ is not acyclic } \}$

  if $\Gamma_t \neq \varnothing$ **then**

    choose $(P, f) \in \Gamma_t$ with probability $f / \left(\sum_{(P_j, f_j) \in \Gamma_t} f_j\right)$

    if $P_{|P|-1} \notin \hat{V}_{\mathcal{T}}$ **then**

      set $U \leftarrow U + P_{|P|-1}$ **and** $\hat{V}_{\mathcal{T}} \leftarrow \hat{V}_{\mathcal{T}} + P_{|P|-1}$

    set $\hat{V}_{\mathcal{T}} \leftarrow \hat{V}_{\mathcal{T}} + t$ **and** $\hat{E}_{\mathcal{T}} \leftarrow \hat{E}_{\mathcal{T}} + (t, P_{|P|-1})$ **and** $\hat{\pi}(t, P_{|P|-1}) \triangleq P$

    set $u(e) \leftarrow u(e) - 1$ **for all** $e \in P$

set $u(e) \leftarrow 0$ **for all** $e = (s, \mathrm{o}_S^-) \in E_{\text{ext}}^{S^-}$ **with** $s \in S \wedge s \notin \hat{V}_{\mathcal{T}}$

set $\bar{T} \triangleq (T \backslash \hat{V}_{\mathcal{T}}) \cup (\{s \in S \cap \hat{V}_{\mathcal{T}} | \delta^+_{\hat{E}_{\mathcal{T}}}(s) = 0\})$

for $t \in \bar{T}$ **do**

  choose $P \leftarrow \texttt{ShortestPath}\left(G^u_{\text{ext}}, c_E, t, \{\mathrm{o}_S^-, \mathrm{o}_r^-\}\right)$
  such that $(\hat{V}_{\mathcal{T}} + t, \hat{E}_{\mathcal{T}} + (t, P_{|P|-1}))$ is acyclic

  if $P = \varnothing$ **then**

    return `null`

  set $\hat{V}_{\mathcal{T}} \leftarrow \hat{V}_{\mathcal{T}} + t$ **and** $\hat{E}_{\mathcal{T}} \leftarrow \hat{E}_{\mathcal{T}} + (t, P_{|P|-1})$ **and** $\hat{\pi}(t, P_{|P|-1}) \triangleq P$

  set $u(e) \leftarrow u(e) - 1$ **for all** $e \in P$

for $e \in \hat{E}_{\mathcal{T}}$ **do**

  set $P \triangleq \hat{\pi}(e)$

  set $\hat{\pi}(e) \leftarrow \langle P_1, \ldots, P_{|P|-1} \rangle$

set $\hat{\mathcal{T}}_G \triangleq$ Virtual Arborescence $(\hat{V}_{\mathcal{T}}, \hat{E}_{\mathcal{T}}, r, \hat{\pi})$

return `PruneSteinerNodes(`$\hat{\mathcal{T}}_G$`)`

8.2. Algorithm PartialDecompose

A major disadvantage of Algorithm FlowDecoRound that was presented in the above section is that paths are selected at random without taking into account our theoretical results on how flow needs to be decomposed (see Section 5.2). Hence, it is rather likely that Algorithm FlowDecoRound disconnects nodes that woubld be connected in an optimal solution.

To alleviate this problem to some extent, we introduce the notion of partial virtual arbores-

Algorithm 8.2: PartialDecompose

Input : Network $G = (V_G, E_G, c_E, u_E)$, Request $R_G = (r, S, T, u_r, c_S, u_S)$, $(\hat{x}, \hat{f}) \in \{(x, f) \in \{0,1\}^S \times \mathbb{Z}_{\geqslant 0}^{E_{\text{ext}}}\}$

Output: Partial Virtual Arborescence $\hat{\mathcal{T}}_G^P$

set $\hat{S} \triangleq \{s \in S | x_s \geqslant 1\}$ **and** $\hat{T} \triangleq T$
set $\hat{\mathcal{T}}_G^P \triangleq (\hat{V}_{\mathcal{T}}^P, \hat{E}_{\mathcal{T}}^P, r, \hat{\pi}^P)$ **where** $\hat{V}_{\mathcal{T}}^P \triangleq \{r\} \cup \hat{S} \cup \hat{T}$, $\hat{E}_{\mathcal{T}}^P \triangleq \varnothing$ **and** $\hat{\pi}^P : \hat{E}_{\mathcal{T}} \to \mathcal{P}_G$
while $\hat{T} \neq \varnothing$ **do**
  let $t \in \hat{T}$ **and** $\hat{T} \leftarrow \hat{T} - t$
  choose $P \triangleq \langle \mathrm{o}^+, t, \ldots, \mathrm{o}_r^- \rangle \in G_{\text{ext}}^{\hat{f}}$
  if $P = \varnothing$ **then**
    goto 3
  end
  for $j = 1$ **to** $|P| - 1$ **do**
    set $\hat{f}(P_j, P_{j+1}) \leftarrow \hat{f}(P_j, P_{j+1}) - 1$
    if Constraint IP-2 is violated *with respect to* $\hat{f}$ and $\hat{S}$ **then**
      choose $W \subseteq V_G$ **such that** $W \cap \hat{S} \neq \varnothing$ **and** $\hat{f}(\delta^+_{E_{\text{ext}}^R}(W)) = 0$
      choose $P' \triangleq \langle P_j, \ldots, \mathrm{o}_S^- \rangle \in G_{\text{ext}}^{\hat{f}}$ **such that** $P_i \in W$ for $1 \leqslant i < m$
      if $P' = \varnothing$ **then**
        set $\hat{f}(P_k, P_{k+1}) \leftarrow \hat{f}(P_k, P_{k+1}) + 1$ **for all** $k \in \{1, \ldots, j\}$
        goto 3
      end
      set $\hat{f}(P_j, P_{j+1}) \leftarrow \hat{f}(P_j, P_{j+1}) + 1$ **and** $\hat{f}(P'_1, P'_2) \leftarrow \hat{f}(P'_1, P'_2) - 1$
      set $P \leftarrow \langle P_1, \ldots, P_{j-1}, P_j = P'_1, P'_2, \ldots, P'_m \rangle$
    end
  end
  if $P_{|P|} = \mathrm{o}_S^-$ **and** $\hat{f}(P_{|P|-1}, P_{|P|}) = 0$ **then**
    set $\hat{S} \leftarrow \hat{S} - P_{|P|-1}$ **and** $\hat{x}(P_{|P|-1}) \leftarrow 0$ **and** $\hat{T} \leftarrow \hat{T} + P_{|P|-1}$
  end
  set $\hat{E}_{\mathcal{T}}^P \leftarrow \hat{E}_{\mathcal{T}}^P + (t, P_{|P|-1})$ **and** $\hat{\pi}^P(t, P_{|P|-1}) \triangleq \texttt{simplify}(\langle P_2, \ldots, P_{|P|-1} \rangle)$
end
return $\hat{\mathcal{T}}_G^P$

cences and give a simple extension of Algoritm Decompose that tries to decompose integral solutions $(\hat{x}, \hat{f}) \in \{(x, f) \in \{0,1\}^S \times \mathbb{Z}_{\geqslant 0}^{E_{\text{ext}}}\}$ as best as possible.

Definition 8.2: PARTIAL VIRTUAL ARBORESCENCE
A partial virtual arborescence $\mathcal{T}_G^P = (V_\mathcal{T}^P, E_\mathcal{T}^P, r, \pi^P)$ is a generalization of virtual arborescences to directed forests, where the connectivity constraint VA-1 of Definition 2.2 is relaxed in the following way. Instead of enforcing VA-1, we only require that there exist a feasible virtual arborescence $\mathcal{T}_G = (V_\mathcal{T}, E_\mathcal{T}, r, \pi)$, such that $V_\mathcal{T}^P = V_\mathcal{T}$, $E_\mathcal{T}^P \subseteq E_\mathcal{T}$ and $\pi(e) \triangleq \pi^P(e)$ for all $e \in E_\mathcal{T}^P$.

Algorithm PartialDecompose (see Algorithm 8.2) is a simple extension of Algorithm Decompose (see Algorithm 5.1 in Section 5.2). The only difference to the original algorithm are the if statements in Lines 6-8 and Lines 14-17 and the different types of the input and the output.

As we do not require the constraints of IP-A-CVSAP to hold for the input $(\hat{x}, \hat{f})$, the **choose** operations in Lines 5 and in Lines 13 may fail at finding or extending a path, that does not violate the connectivity of other active Steiner nodes. In case one of these **choose** operations fails, the path construction is aborted and all decrementations are revoked. The algorithm then continues by trying to connect another unconnected terminal.

To see that Algorithm PartialDecompose yields indeed a *partial* virtual arborescence, it only needs to be shown that $(\hat{V}_\mathcal{T}^P, \hat{E}_\mathcal{T}^P)$ is acyclic. This however holds by the same argument that was used in the proof of Theorem 5.8: a Steiner node is only placed in the set of unconnected nodes $\hat{T}$ once, when all its incoming connections have been removed. Therefore Algorithm PartialDecompose indeed outputs a partial virtual arborescence.

8.3. Algorithm Virtual Capacitated Prim Connect

In this section the Algorithm VCPrimConnect (see Algorithm 8.3) is introduced that tries to connect a partial virtual arborescence to obtain a feasible solution to the corresponding CVSAP instance. The algorithm relies on the following important observation:

Observation 8.3: If all active Steiner nodes are connected using fixed paths, then terminals can be optimally connected using algorithm `MinCostAssignment`, if such an assignment exists.

Algorithm VCPrimConnect tries to exploit this fact by first connecting all active Steiner nodes that are contained in a partial virtual arborescence $\mathcal{T}_G^P$ and then computing an optimal assignment of unconnected terminals towards the set of Steiner nodes and the root.

Initially, the set of unconnected nodes U and the set of unconnected Steiner nodes $\bar{S}$ are computed (see Lines 1 and 2). In Line 3, local copies of the node set, edge set and of the function π of the partial virtual arborescence $\mathcal{T}_G^P$ are created. Next the currently remaining capacity $u : E_G \to \mathbb{N}$ is computed in Line 4.

Algorithm 8.3: VCPrimConnect

Input : Network $G = (V_G, E_G, c_E, u_E)$, Request $R_G = (r, S, T, u_r, c_S, u_S)$, *Partial* Virtual Arborescence $\mathcal{T}_G^P = (V_\mathcal{T}^P, E_\mathcal{T}^P, r, \pi^P)$
Output: Feasible Virtual Arborescence $\mathcal{T}_G = (V_\mathcal{T}, E_\mathcal{T}, r, \pi)$ or `null`

1 **set** $U \triangleq \{v | v \in V_\mathcal{T}^P \backslash \{r\}, \delta^+_{E_\mathcal{T}^P}(v) = 0\}$
2 **set** $\bar{S} \triangleq U \cap S$
3 **set** $V_\mathcal{T} \triangleq V_\mathcal{T}^P$, $E_\mathcal{T} \triangleq E_\mathcal{T}^P$ **and** $\pi(u, v) = \pi^P(u, v)$ **for all** $(u, v) \in E_\mathcal{T}$
4 **set** $u(e) \triangleq u_E(e) - |\pi(E_\mathcal{T})[e]|$ **for all** $e \in E_G$
5 **while** $\bar{S} \neq \varnothing$ **do**
6 **compute** $R \leftarrow \{r' | r \in \{r\} \cup (V_\mathcal{T} \cap S), r'$ reaches r in $\mathcal{T}_G, \delta^-_{E_\mathcal{T}}(r') < u_{r,S}(r')\}$
7 **compute** $P = \texttt{MinAllShortestPath}(G^u, c_E, \bar{S}, R)$
8 **if** $P = \texttt{null}$ **then**
9 **return** `null`
10 **end**
11 **set** $\bar{S} \leftarrow \bar{S} - P_1$
12 **set** $E_\mathcal{T} \leftarrow E_\mathcal{T} + (P_1, P_{|P|})$ **and** $\pi(P_1, P_{|P|}) \triangleq P$
13 **set** $u(e) \leftarrow u(e) - 1$ **for all** $e \in P$
14 **end**
15 **set** $\bar{T} \triangleq U \cap T$
16 **set** $u_V(r') \triangleq u_{r,S}(r') - \delta^-_{E_\mathcal{T}}(r')$ **for all** $r' \in \{r\} \cup (V_\mathcal{T} \cap S)$
17 **compute** $\Gamma = \{P^{\bar{t}}\} \leftarrow \texttt{MinCostAssignment}(G, c_E, u, u_V, \bar{T}, \{r\} \cup V_\mathcal{T} \cap S)$
18 **if** $\Gamma = \varnothing$ **then**
19 **return** `null`
20 **end**
21 **set** $E_\mathcal{T} \leftarrow E_\mathcal{T} + (t, P^t_{|P^t|})$ **and** $\pi(t, P^t_{|P^t|}) \triangleq P^t$ **for all** $P^t \in \Gamma$
22 **return** $\mathcal{T}_G \triangleq (V_\mathcal{T}, E_\mathcal{T}, r, \pi)$

Lines 5-14 constitute the first phase of the algorithm, in which Steiner nodes are connected by a simple adaptation of the algorithm by Prim to compute minimal spanning trees [KV12]. The shortest path from the set of unconnected Steiner nodes to the connected component of $(V_\mathcal{T}, E_\mathcal{T})$ that contains the root is computed, such that nodes that have no capacity left are again excluded (see Lines 6 and 7). To shorten notation, we denote by $u_{r,S}$ the extension of u_S on the set $\{r\} \cup S$, such that $u_{r,S}(r) = u_r$ holds. If no such shortest path P exists, then the algorithm aborts and returns `null`. However, if the shortest path P exists, it is included in the solution and the edge capacities are adapted accordingly (see Lines 11-13). Therefore, if Line 15 is reached, all active Steiner nodes connect to the root r in $(V_\mathcal{T}, E_\mathcal{T})$.

In Lines 15-21 the unconnected terminals $\bar{T}$ are connected (if possible) using algorithm `MinCostAssignment`. As a preparation for the call to `MinCostAssignment`, in Line 15 the remaining node capacities are stored in the function u_V. If no assignment Γ was found, then `null` is returned. Otherwise each unconnected terminal $\bar{t} \in \bar{T}$ is connected according to the found path $P^{\bar{t}} \in \Gamma$ in Line 21 and the feasible virtual arborescence $\mathcal{T}_G$ is returned.

It is obvious that if a solution is returned, Algorithm VCPrimConnect extends a partial virtual arborescence to a feasible solution and we only shortly prove the polynomial runtime in the following theorem:

Theorem 8.4: ***Runtime of Algorithm VCPrimConnect***

The runtime of Algorithm VCPrimConnect is bounded by $\mathcal{O}(|V_G| \cdot |E_G| + |V_G|^2 \log |V_G|)$.

Proof: In the first phase when Steiner nodes are connected, the while loop is executed at most $|\bar{S}| \leqslant |V_G|$ many times. Within the loop the computationally most expensive task is computing the shortest path in Line 7. As the runtime of computing this shortest path is bounded by $\mathcal{O}(|E_G| + |V_G| \log |V_G|)$ (see Remark 6.4),the runtime of the first is bounded by $\mathcal{O}(|V_G| \cdot |E_G| + |V_G|^2 \log |V_G|)$. As the runtime of the remaining code is dominated by the call to `MinCostAssignment` and as the bound of the runtime of `MinCostAssignment` (see Remark 6.8) equals the runtime bound obtained for the first phase, the theorem is proven. ■

8.4. Abstract Interface to LP Solver

The heuristics presented in Sections 8.5 and 8.6 will actively solve linear relaxations after having introduced local constraints. We will use the following abstract functions. The function `solveSeparateSolve`() solves the LP and then separates the connectivity inequalities IP-2 of IP-A-CVSAP. If violated cuts have been found, these are added to the current node and the LP is resolved. Note that the separation procedure is only called once such that only some of the violated connectivity inequalities may have been found. The function `solveSeparateSolve` returns the optimal LP solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{LP}}$ to the current problem.

We assume that the solver abstractly provides the following functions to query the solver's state. The function `infeasibleLP`() returns as boolean value whether solving the LP was aborted due to the LP's infeasibility. The function `objectiveLimit`() returns as boolean value, whether solving the LP was aborted as it was proven that the objective value of the LP is larger than the *primal bound*, which is given by the objective value of the best known solution. As e.g. our diving heuristics presented in Section 8.5 strictly rely on the LP's solution, these heuristics abort once the primal bound is exceeded as they will not be able to find a cheaper solution. On the other hand, for the heuristics presented in Section 8.6 the LP solution value will not be a dual bound for the solutions constructed. We therefore allow to disable the primal bound via the function call `disableGlobalPrimalBound`().

To add *local* constraints the function `addConstraintsLocally`($\mathcal{C}$) will be used, that takes as input a set $\mathcal{C}$ of linear (in)equalities on variables already contained in the current problem formulation. Note that local constraints only apply during the execution of the respective heuristics but are not lifted to the general problem formulation when executed as part of a branch-and-bound solver.

8.5. Greedy Diving Heuristics

In this section we present a diving heuristic and two derived heuristics (see [Ach07] for an introduction to diving heuristics). The common approach taken by diving heuristics is to iteratively fix a subset of variables and to recompute the linear relaxation, to obtain an integral feasible solution. As shown in the computational evaluation, the diving heuristics implemented by SCIP may fail to produce solutions. We believe this to be due to the fact, that in SCIP's heuristics no separation procedures for the connectivity inequalities are performed and as they cannot handle infeasibilities by means other than backtracking.

The diving heuristic presented henceforth will first *iteratively* fix all Steiner site variables and then iteratively fix all edge variables, to (try to) obtain a feasible solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$. After each iteration the LP is resolved and connectivity inequalities IP-2 of IP-A-CVSAP are separated, possibly necessitating the resolving of the LP. As these operations are computationally expensive, our heuristics are designed according to the following principles.

1. To avoid infeasibilities as best as possible, only lower bounds are increased such that either Steiner nodes are activated or the flow on an edge is increased.

2. Instead of performing backtracking once LP infeasibilities occur, the heuristics try to use the last feasible solution to construct a solution.

We will now give an overview over Algorithm GreedyDiving and in Section 8.5.2 derive two other heuristics from it. In Section 8.7 the runtime of all lp-based heuristics will be discussed.

8.5.1. Synopsis of Algorithm GreedyDiving

Algorithm GreedyDiving takes as main input the LP solution of the current node in the branch-and-bound tree or the root relaxation. Initially all Steiner nodes which are activated less than 1% or more than 99% are fixed to be deactivated or are fixed to be activated respectively (see Lines 1 and 2). Next the sets $\dot{S}, \dot{E}$ are defined which are used to keep track of the Steiner nodes whose opening variables are fixed and of the edges whose flow variables have been fixed (see Line 3).

Within the main loop (starting in Line 4) additional variable fixings will be introduced, after (re)solving the LP in Line 5. If the LP could not be solved due to infeasibilities (see Line 7), but all Steiner nodes variables are fixed, the main loop is exited to try to construct a solution in Lines 31-33. However, if infeasibilities arise before all Steiner site variables are fixed or if the objective limit was reached, the algorithm aborts by returning `null`. As we do not perform backtracking, each linear realaxation's objective is a lower bound for the solution generated. Exiting the heuristic as soon as the objective limit is reached, therefore allows to not spent further time in searching for a better solution that cannot be obtained by this approach.

In Line 10 the local solution variables $(\hat{x}, \hat{f})$ are overwritten by the LP's last solution, such that if infeasibilities occured, the last feasible solution will still be stored in $(\hat{x}, \hat{f})$ if the main loop was left in Line 30. If not all Steiner site variables were fixed, in Lines 12-19 additional Steiner site variable fixings are introduced. First, all Steiner sites which are activated less than 1% or more than 99% are deactivated or activated. In Lines 17-19 at least a single Steiner site

is activated, if previously not all variables were fixed. We opt to open the Steiner site with the smallest installation costs divided by its opening variable. Therefore, under equal Steiner costs, the node with that is opened the most will be selected to be opened.

Similarly, if all Steiner site variables were fixed beforehand, in Lines 21-24 local fixings are applied to edge variables. In contrast to Steiner nodes, we fix edge variables initially only if they are within $1‰$ of the next integer (see Lines 21-22). If the flow variables of some edges have not been fixed, the single edge $\hat{e}$ whose flow value is closest to its next larger integer is selected and the corresponding lower bound is introduced (see Lines 26-28).

If $\dot{S} = S$ and $\dot{E} = E_{\text{ext}}$ holds, the Algorithm GreedyDiving exits the diving (see Lines 30) after having resolved the LP and stored the solution in $(\hat{x}, \hat{f})$.

To finally construct a solution, first all unfixed flow variables are simply rounded down (see Line 31) and the Algorithm PartialDecompose is called to obtain a partial virtual arborescence $\hat{\mathcal{T}}_G^P$ in Line 32. Note that if an integral solution was found, rounding flow variables will have no effect. Furthermore, if for the found solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{IP}}$ holds, then Algorithm PartialDecompose will construct indeed a feasible virtual arborescence. If a feasible virtual arborescence was constructed, Algorithm will have no effect and the solution is returned unchanged.

However, in case that Algorithm PartialDecompose did fail to connect all nodes, Algorithm VCPrimConnect will first (try to) connect all unconnected activated Steiner nodes and then assign still unconnected terminals.

8.5.2. Derived Variants

Since especially fixing all edge variables can be potentially very time consuming, we also consider the following variant of Algorithm GreedyDiving.

Definition 8.5: ALGORITHM GREEDYSTEINERDIVING
The algorithm SteinerGreedyDiving is the variant of Algorithm GreedyDiving, in which diving is only performed on Steiner node variables, such that Lines 20-28 are removed from Algorithm GreedyDiving altogether.

Note that even though algorithm SteinerGreedyDiving does not perform diving on flow variables, it may obtain an integral solution. In any case, if at least a large fraction of flow variables was integral, applying Algorithm PartialDecompose can increase the chances that Algorithm VCPrimConnect can connect all remaining unconnected nodes.

To further reduce the number of LP computations and separation rounds used, we also consider the following variant of the GreedySteinerDiving algorithm.

Definition 8.6: ALGORITHM FASTGREEDYSTEINERDIVING
The algorithm FastSteinerGreedyDiving is the variant of GreedySteinerDiving, in which in Line 17 not a single node, but the 5% of the nodes contained in $\hat{S}$ are selected and fixed to be opened, which achieve the best ratio $c_S(s)/\hat{x}_s$.

Algorithm 8.4: GreedyDiving

Input : Network $G = (V_G, E_G, c_E, u_E)$, Request $R_G = (r, S, T, u_r, c_S, u_S)$, LP relaxation solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{LP}}$ to IP-A-CVSAP

Output: A Feasible Virtual Arborescence $\hat{\mathcal{T}}_G$ or `null`

set $\lfloor S \rfloor \triangleq \{s \in S | \hat{x}_s \leqslant 0.01\}$ **and** $\lceil S \rceil \triangleq \{s \in S | \hat{x}_s \geqslant 0.99\}$
`addConstraintsLocally`$(\{x_s = 0 | s \in \lfloor S \rfloor\} \cup \{x_s = 1 | s \in \lceil S \rceil\})$
set $\dot{S} \triangleq \lfloor S \rfloor \cup \lceil S \rceil$ **and** $\dot{E} \triangleq \varnothing$
do
  $(\hat{x}', \hat{f}') \leftarrow$ `solveSeparateSolve()`
  if *infeasibleLP()* ***and*** $\dot{S} = S$ **then**
    break
  else if *infeasibleLP()* ***or*** *objectiveLimit()* **then**
    return *null*
  set $(\hat{x}, \hat{f}) \leftarrow (\hat{x}', \hat{f}')$
  if $\dot{S} \neq S$ **then**
    set $\lfloor S \rfloor \triangleq \{s \in S | \hat{x}_s \leqslant 0.01\}$ **and** $\lceil S \rceil \triangleq \{s \in S | \hat{x}_s \geqslant 0.99\}$
    `addConstraintsLocally`$(\{x_s = 0 | s \in \lfloor S \rfloor\} \cup \{x_s = 1 | s \in \lceil S \rceil\})$
    set $\dot{S} \leftarrow \dot{S} \cup \lfloor S \rfloor \cup \lceil S \rceil$
    set $\hat{S} \triangleq S \backslash \dot{S}$
    if $\hat{S} \neq \varnothing$ **then**
      choose $\hat{s} \in \hat{S}$ **with** $c_S(\hat{s})/\hat{x}_{\hat{s}}$ **minimal**
      `addConstraintsLocally`$(\{x_{\hat{s}} = 1\})$
      set $\dot{S} \leftarrow \dot{S} + \hat{s}$
  else if $\dot{E} \neq E_{\text{ext}}$ **then**
    set $\lfloor E \rfloor \triangleq \{e \in E_{\text{ext}} | \, |\hat{f}_e - \lfloor \hat{f}_e \rfloor| \leqslant 0.001\}$, $\lceil E \rceil \triangleq \{e \in E_{\text{ext}} | \, |\hat{f}_e - \lceil \hat{f}_e \rceil| \leqslant 0.001\}$
    `addConstraintsLocally`$(\{f_e = \lfloor \hat{f}_e \rfloor | e \in \lfloor E \rfloor\} \cup \{f_e = \lceil \hat{f}_e \rceil | e \in \lceil E \rceil\})$
    set $\dot{E} \leftarrow \dot{E} \cup \lfloor E \rfloor \cup \lceil E \rceil$
    set $\hat{E} \triangleq E_{\text{ext}} \backslash \dot{E}$
    if $\hat{E} \neq \varnothing$ **then**
      choose $\hat{e} \in \hat{E}$ **with** $\lceil \hat{f}_{\hat{e}} \rceil - \hat{f}_{\hat{e}}$ **minimal**
      `addConstraintsLocally`$(\{\hat{f}_{\hat{e}} \geqslant \lceil \hat{f}_{\hat{e}} \rceil\})$
      set $\dot{E} \leftarrow \dot{E} + \hat{e}$
  else
    break
set $\hat{f}_e \leftarrow \lfloor \hat{f}_e \rfloor$ **for all** $e \in E_{\text{ext}} \backslash \dot{E}$
set $\hat{\mathcal{T}}_G^P \leftarrow$ `PartialDecompose` $(G, R_G, (\hat{x}, \hat{f}))$
return `VCPrimConnect` $(G, R_G, \hat{\mathcal{T}}_G^P)$

It must be noted that we have chosen the bound of 5% rather arbitrarily. Of importance as a design decision is to relate the number of nodes to activate to the number of unfixed Steiner nodes. Since the number of Steiner nodes decreases in each iteration, initially more Steiner nodes will be opened than in later iterations, such that with each iteration the selection process becomes more selective.

8.6. Multiple Shots Heuristics

The heuristics presented in the above section require all Steiner site variables to be fixed to obtain a solution. To circumvent this and to obtain solutions faster, we present the Algorithm MultipleShots which interprets Steiner site opening variables as probabilities, opens Steiner sites according to these probabilities, and tries to construct a solution by only using the selected Steiner nodes. This procedure is iterated and combined with resolving the LP and separating inequalities to obtain *accurate* 'probabilities' in each iteration.

The next section provides a detailed synopsis of Algorithm MultipleShots and in Section 8.6.2 a simple variant will be derived.

8.6.1. Synopsis of Algorithm MultipleShots

As in Algorithm GreedyDiving initially all nodes that either activated less than 1% or more than 99% are activated. The algorithm will keep track of the Steiner nodes that have been activated (set $\dot{S}_1$) as well as of the Steiner nodes that have been deactivated (set $\dot{S}_0$); these sets are initialized in Line 3. Importantly, in Line 4 the primal bound is (locally) disabled, such that the solver will not abort solving the LP once the primal bound has been exceeded. This is important, as the node selection might introduce too many Steiner nodes that are not necessary to obtain a solution and Algorithm PruneSteinerNodes is applied if a solution has been found, such that the objective value might *decrease*.

After having recomputed the LP and set $\dot{S}_0$ and $\dot{S}_1$ accordingly in Lines 6-10, in Lines 12-20 a subset of unfixed Steiner nodes $S_1 \subset \hat{S}$ is selected to be opened according to the Steiner sites' activation values $\hat{x}_s$. After having decided which Steiner sites to open, the Algorithm VCPrimConnect is called *in each iteration* to try to find a feasible solution using only the Steiner sites contained in $\dot{S}_1$ (see Lines 21 and 22). If indeed a solution was found, active Steiner nodes are pruned by calling Algorithm PruneSteinerNodes and the resulting feasible virtual arborescence is returned in Line 24.

Lastly note that if - by chance - the set S_1 of selected Steiner sites to be opened is empty after executing Line 15 and only less than ten Steiner nodes are unfixed, all these Steiner sites are activated in Line 16 and 17. While the choice of 'ten' is again rather arbitrary, we therefore forbid possibly very long loops for deciding which Steiner sites to open. However, even without this mechanism, the expected number of iterations needed for selecting the Steiner sites to open, is bounded by the constant 100 as $\hat{S}$ contains at least a single element and all elements have an opening 'probability' larger than 1%. The Lines 16,17 will therefore be rather of use for the simple variant which is presented in the next section, as it additionally scales down opening probabilities.

Algorithm 8.5: MultipleShots

Input : Network $G = (V_G, E_G, c_E, u_E)$, Request $R_G = (r, S, T, u_r, c_S, u_S)$, LP relaxation solution $(\hat{x}, \hat{f}) \in \mathcal{F}_{\text{LP}}$ to IP-A-CVSAP

Output: A Feasible Virtual Arborescence $\hat{\mathcal{T}}_G$ or `null`

set $\lfloor S \rfloor \triangleq \{s \in S | \hat{x}_s \leqslant 0.01\}$ **and** $\lceil S \rceil \triangleq \{s \in S | \hat{x}_s \geqslant 0.99\}$
`addConstraintsLocally`$(\{x_s = 0 | s \in \lfloor S \rfloor\} \cup \{x_s = 1 | s \in \lceil S \rceil\})$
set $\dot{S}_0 \triangleq \lfloor S \rfloor \cup$ **and** $\dot{S}_1 \triangleq \lceil S \rceil$
`disableGlobalPrimalBound`()
repeat
  $(\hat{x}, \hat{f}) \leftarrow$ `solveSeparateSolve`()
  if `infeasibleLP`() **return** *null*
  set $\lfloor S \rfloor \triangleq \{s \in S | \hat{x}_s \leqslant 0.01\}$ **and** $\lceil S \rceil \triangleq \{s \in S | \hat{x}_s \geqslant 0.99\}$
  `addConstraintsLocally`$(\{x_s = 0 | s \in \lfloor S \rfloor\} \cup \{x_s = 1 | s \in \lceil S \rceil\})$
  set $\dot{S}_0 \leftarrow \dot{S}_0 \cup \lfloor S \rfloor$ **and** $\dot{S}_1 \leftarrow \dot{S}_1 \cup \lceil S \rceil$
  set $\hat{S} \triangleq S \backslash (\dot{S}_0 \cup \dot{S}_1)$
  if $\hat{S} \neq \varnothing$ **then**
    repeat
      set $S_1 \triangleq \hat{S}$
      remove s **from** S_1 **with probability** $1 - \hat{x}_s$ **for all** $s \in S_1$
      if $S_1 = \varnothing$ ***and*** $|S \backslash (\dot{S}_0 \cup \dot{S}_1)| < 10$ **then**
        set $S_1 \leftarrow S \backslash (\dot{S}_0 \cup \dot{S}_1)$
    until $S_1 \neq \varnothing$
    `addConstraintsLocally`$(\{x_s = 1 | s \in S_1\})$
    set $\dot{S}_1 \leftarrow \dot{S}_1 \cup S_1$
  $\hat{\mathcal{T}}_G^P \triangleq (\hat{V}_{\mathcal{T}}^P, \hat{E}_{\mathcal{T}}^P, r, \varnothing)$ **where** $\hat{V}_{\mathcal{T}}^P \triangleq \{r\} \cup T \cup \dot{S}_1$ **and** $\hat{E}_{\mathcal{T}} \triangleq \varnothing$
  set $\hat{\mathcal{T}}_G \triangleq$`VCPrimConnect`$(G, R_G, \hat{\mathcal{T}}_G^P)$
  if $\hat{\mathcal{T}}_G \neq$ *null* **then**
    return `PruneSteinerNodes`$(\hat{\mathcal{T}}_G)$
until $\dot{S}_0 \cup \dot{S}_1 = S$
return *null*

8.6.2. Variant MultipleShotsSquared

The approach taken by Algorithm MultipleShots is similar to the one employed in the fast greedy diving algorithm presented in Section 8.5.2, as in each iteration a set of Steiner sites is activated. However, in the fast greedy diving algorithm the number of activated Steiner sites is strictly bounded to be less than 5%. In contrast, Algorithm MultipleShots can activate an arbitrary amount of Steiner sites. Even though the Steiner site opening variables can be naturally interpreted as probabilities, chances are that Algorithm MultipleShots activates too many Steiner nodes in a single iteration. We therefore propose the following simple variant

which scales down opening probabilities by squaring them.

Definition 8.7: ALGORITHM MULTIPLESHOTSSQUARED
The algorithm MultipleShotsSquared is the variant of Algorithm MultipleShots, in which in Line 15 each node is removed with probability $1 - \hat{x}_s^2$.

8.7. Runtime Considerations

In this section we conclude our discussion of the linear programming based heuristics presented by showing that these are indeed polynomial algorithms.

First of all note that linear programming formulations can be solved in polynomial time by using e.g. the ellipsoid method (see [MG07] for a nice introduction). Furthermore, Grötschel, Lovász and Schrijver have the equivalence of strong optimization and separation [GLS]. By this famous theorem the linear relaxation of IP-A-CVSAP can be solved within polynomial time as there exists a strong separation oracle for the set of exponential constraints IP-2.

While the above results are very important, linear relaxations are still in practice mostly solved using the simplex algorithm, even though the polynomial runtime of the simplex algorithm could not be established [MG07].

As our solver (see [RS13a]) relies on the simplex algorithm to solve linear relaxations and runtime bounds to solve linear relaxations in polynomial time are not practical, we only argue that the heuristics defined are indeed polynomial but do not give upper bounds on the runtime.

To check that Algorithm GreedyDiving can be implemented in polynomial time, note that the main loop will be executed at most $\mathcal{O}(|V_G| \cdot |E_G|)$ many times, as in each iteration either a Steiner site variable is fixed or the lower bound of a flow variable is increased. As $u_E(e) \leqslant |V_G|$ can be assumed without loss of generality, after $\mathcal{O}(|V_G| \cdot |E_G|)$ many iterations all variables are fixed and the loop is exited. As in each iteration the linear relaxation for which all constraints, including all connectivity inequalities, hold, can be computed in polynomial time by the result of Grötschel et al, the overall runtime of Lines 4-30 is polynomially bounded. As the polynomial runtime of Algorithm VCPrimConnect has been established in Theorem 8.4 and the polynomial runtime of Algorithm PartialDecompose follows from Lemma 5.12, we obtain the following result.

Theorem 8.8: ***The greedy diving heuristics can be implemented to run in polynomial time.***

As above the polynomial runtime of Algorithm GreedyDiving and the same considerations also apply to its variants, these are all indeed polynomial time heuristics.

The polynomial runtime of the multiple shots heuristics follows by the same argument, as in each iteration at least a single Steiner site variable is fixed in constant expected time (see the discussion at the end of Section 8.6.1) and all Algorithm VCPrimConnect has a polynomial runtime.

Corollary 8.9: The Multiple Shots Heuristics can be implemented to run in polynomial time. ■

Part IV.

Computational Evaluation

9. Outline of the Computational Evaluation

In Part II two distinct exact algorithms have been proposed for solving CVSAP, namely the naive multi-commodity flow formulation A-CVSAP-MCF and the VirtuCast Algorithm that builds upon the single-commodity flow formulation IP-A-CVSAP. In part III four distinct types of heuristics have been presented to (try to) compute feasible solutions for CVSAP, namely the combinatorial Algorithm GreedySelect and the LP-based algorithms FlowDeco-Round, GreedyDiving and MultipleShots.

In this part of the thesis, all of the above algorithms are evaluated in an extensive *explorative* study on three distinct topologies (see Section 10). For each topology five different graph sizes with 15 instances each are considered. The instances per graph size are generated subject to different cost and capacity distributions, to allow for evaluating the algorithms' general peformance. Since only 15 instances are considered, the performance of the algorithms will not be discussed with respect to specific cost or capacity distributions. Nevertheless, by considering 225 instances overall, the computational evaluation does not only explore the performance but also allows to draw *qualitative* conclusions.

As VirtuCast is a branch-and-cut algorithm, in Section 11 first a set of separation parameters and branching rules are chosen to obtain the best possible dual bounds. In Section 12 the performance of the LP-based heuristics is assessed within the VirtuCast branch-and-cut solver with regard to the following criteria: efficiency in finding solutions, the quality of the solutions found and the runtime. Having established general observations, the topology-dependent performance is considered, yielding a selection of (topology-dependent) heuristics to include as primal heuristics in the final VirtuCast branch-and-cut solver.

While Sections 11 and 12 are necessary preliminary steps to obtain the best overall VirtuCast performance, both sections yield valuable contributions in their own right. With respect to the separation parameters we find that employing *nested cuts* does not improve the performance while employing *creep-flow* is crucial. This contrasts the observations of Koch and Martin [KM98], who established that both are beneficial for separating Steiner cuts. Even more importantly, in Section 12 distinct topology-independent performance characteristics for the proposed LP-based heuristics are obtained.

Having thoroughly discussed separation and branching parameters in Section 11 as well as the performance of heuristics in Section 12, the performance achieved by the final VirtuCast solver is discussed rather shortly in Section 13. Lastly, in Section 14 the performance of the final VirtuCast solver is used as a baseline for the performance evaluation of the following alorithms:

1. the multi-commodity flow formulation A-CVSAP-MCF, which is solved using CPLEX,

2. the combinatorial heuristic GreedySelect and

3. the VirtuCast solver with only SCIP's heuristics enabled.

In the following, Section 9.1 introduces measures used to evaluate the performance of algorithms and Section 9.2 introduces the general computational setup that was used for all experiments.

9.1. Notation & Measures

As we will use 15 instances for each topology and each graph size, we mainly use box plots to present our results. Note that all boxplots presented in this section use the standard $1.5 * IQR$ whiskers of R (and not the 95th and 5th percentiles).

To measure the quality of solutions, the relative objective gap is used, which is computed as $(P - D)/D$ for minimization problems, where P denotes the primal bound, i.e. the objective of the best known solution, and D denotes the dual bound.

Similarly, the improvement of the dual bound is measured as $(D_f - D_r)/D_r$ where D_r denotes the dual bound at the root and D_f denotes the final dual bound.

When measuring the objective gap, we sometimes incorporate *infinite* gaps in the boxplot to denote that no primal solution was found. Within the plots, the ∞ symbol is used.

9.2. General Computational Setup

All our experiments were conducted on machines equipped with an 8-core Intel Xeon L5420 processor running at 2.5 Ghz and 16 GB RAM.

VirtuCast was implemented in C/C++ using the SCIP framework as underlying branch-and-cut framework. The VirtuCast solver allows for the multi-threaded separation of connectivity inequalities via a thread pool that is implemented using fine-grained locking. This feature is always enabled and we use 8 threads corresponding to the number of (physical) cores. Even though the performance improvement by utilizing multi-threading is not studied within this thesis, we note that preliminary experiments have shown a performance improvement even on small instances where only few tasks, i.e. Steiner sites or terminals to compute maximum flows from, exist.

Furthermore, the maximal number of separation rounds is set to 5 for all experiments, such that the LP will at most be resolved at each node 5 times. At the root however, we allow for arbitrary many separation rounds.

As the VirtuCast solver as well as CPLEX, which is used to solve formulation A-CVSAP-MCF, rely on multithreading, all given (run)times are wallclock times.

Lastly, note that all of the considered algorithms have been implemented by the author without utilizing any algorithmic libraries in C/C++.

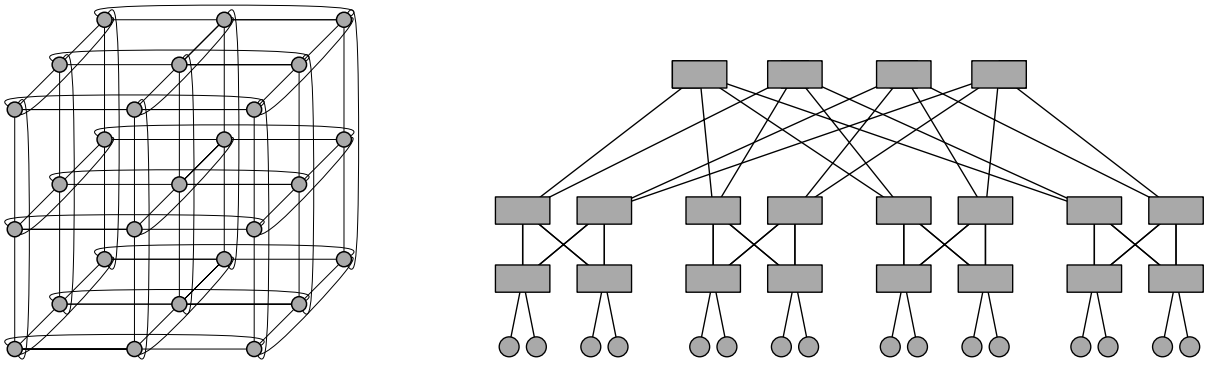
10. Topologies

In this section the different toplogies used in the computational evaluation are presented, namely fat trees, 3D tori and synthetic ISP topologies, which are generated by IGen [Quo+09]. In Section 10.1 a short overview in which contexts these topologies are used is given and their characteristics are discussed. In Section 10.2 an overview over the (common) generation parameters is given and Sections 10.3 to 10.5 present the detailed generation parameters for fat tree, 3D torus and IGen instances respectively.

10.1. Selected Topologies

Fat trees are a scalable data center topology that were introduced in the work of Al-Fares et al. [AFLV08]. Each fat tree network consists of three layers of k-port switches (see Figure 10.1b). The bottom two layers interconnect in a full bipartite fashion and form so called pods. Each pod is connected to several core switches which constitute the top-most layer. Hosts are connected to the lowest layer of switches only. Note that the diameter of this topology is constant, independent of how many hosts are connected. Fat trees have received much attention lately as the bandiwdth available for hosts to send and receive data is only limited by the (single) connection with which the hosts are connected to their pod. Therefore, fat trees are highly connected, aiming at alleviating the common oversubscription of links in data centers [BH09].

3D tori are another type of data center topologies. A 3D torus is a three dimensional grid graph, where each node is connected to exactly six other nodes (see Figure 10.1a). This topology is often used in high performance computing and big data applications [Cos+12]. In



(a) A 3D torus of side length 3.

(b) A fat tree using 4-port switches (rectangles).

Figure 10.1.: Examplary 3D torus and fat tree topologies.

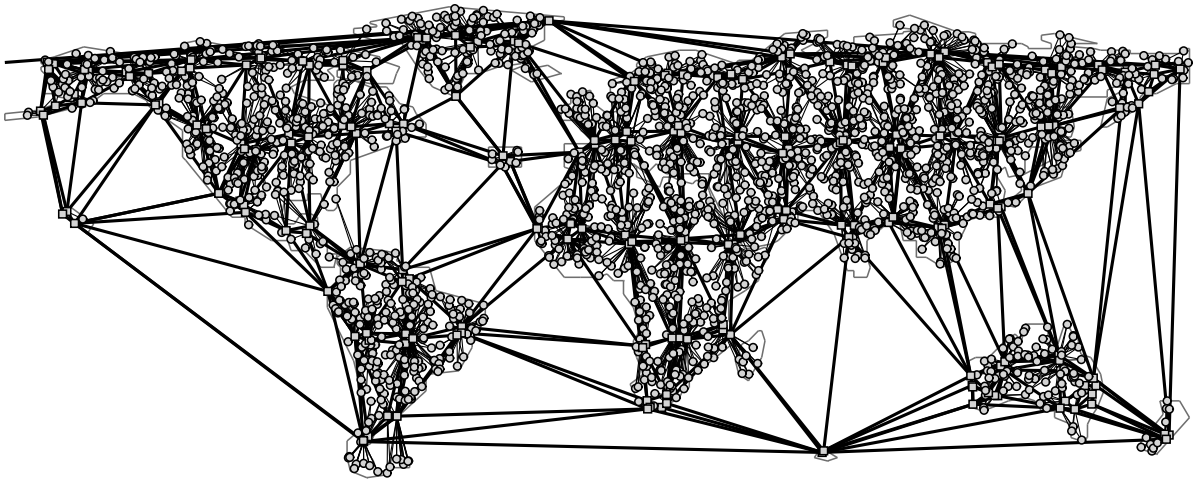


Figure 10.2.: An ISP topology generated by IGen with 2400 nodes.

contrast to fat trees, 3D tori have a limited degree and posess a diameter of $\mathcal{O}(\sqrt[3]{n})$ where n is the number of nodes.

The ISP topologies we consider are synthetic ones generated by the IGen topology generator [Quo+09]. In contrast to the above introduced data center networks, the generated ISP topologies consist of locally well-connected clusters that model autonomous systems (AS), which interconnect via a (planar) backbone. The nodes used to connect different ASes model points of presences (PoPs). Due to the planarity of the backbone, the diameter is generally not bounded. In Figure 10.2 an example topology is depicted.

10.2. Generation Parameters

Even though we do not target specific applications, the following parameters will be chosen topology dependent.

Selection of Steiner sites Given a topology, a subset of nodes needs to be selected to become Steiner sites. As nodes in the network may represent different entities of the topology, this selection is topology dependent.

Steiner site installation costs We will consider only one *type* of Steiner nodes, such that all Steiner sites have the same capacity. Therefore, Steiner site costs will generally be fixed. However, on ISP networks even hosting the same processing functionality might induce different costs at different locations.

Edge costs For the regular 3D tori we will assume fixed unit edge costs. For fat trees, we will assume the connections towards core switches (top most layer) to be more costly than other links, such that processing functionality is more likely to be placed within pods. On IGen instances, edge costs are generally defined by the metric costs of the incident nodes.

The following parameters are common to all topologies.

Steiner site capacities Independent of the topology, we fix the capacity of Steiner sites and the root to be 5. However, as we will vary the costs of opening Steiner sites, the installation cost per incoming connection will vary accordingly. Thus, this has a similar effect as if costs were fixed and capacities are scaled.

Edge capacities Independent of the topology, we will use three types of edge capacity distributions that assign an edge a capacity between one and three. The first one chooses with a probability of 60% an unit capacity, with 30% a capacity of two and with 10% a capacity of 3. For the other two capacity distributions probabilities of 20%, 60%, 20% and 10%, 30%, 60% are used for selecting a capacity of one, two or three. By varying edge capacities, different loads on the topology are modeled.

10.3. Fat Tree

We have selected the graph sizes as listed in Table 10.1, such that fat tree topologies are constructed using $\{8, 12, \ldots 24\}$-port switches. All switches are selected to be possible Steiner sites and half of the possible hosts are uniformly at random selected to be terminals.

We define edge costs between switches in pods to be one towards the core layer to be four. Edges connecting terminals are assigned no costs, as the terminal is only connected via a single edge. Note that given these edge costs, each terminal can reach each other terminal by a path of cost less than or equal to 10. According to this observation, we choose fixed Steiner site installation costs to be either 1, 10, 20, 30 or 40. Therefore, if e.g. the installations costs are 1, a Steiner site will be always activated if more than one flow traverses it. On the other hand, a cost of 40 implies that installing a Steiner node, reduces the overall costs if 5 nodes are connected to it.

Graph Size	Ports	Nodes	Edges	Steiner Sites	Terminals
I	8	112	544	80	32
II	12	288	1836	180	108
III	16	576	4352	320	256
IV	20	1000	8500	500	500
V	24	1584	14680	720	864

Table 10.1.: Graph sizes for fat tree instances.

10.4. 3D Torus

We have selected the graph sizes as listed in Table 10.2. Steiner sites are distributed equally spaced throughout the network, such that a quarter of the nodes may process flows. Terminals

are distributed uniformly at random. Edges have a fixed cost of 1. The installation costs for Steiner sites are chosen to be 0.125, 0.25, 0.5, 1 or 2 times the diameter of the torus, which is $3k/2$ for a side length of k (for k even). The idea behind these Steiner site costs is again to vary the cost point at which the installation of processing functionality can reduce costs. For Steiner costs of 2 times the diameter, the installation of a Steiner node only makes sense if at least 4 flows can be processed, as the Steiner node needs to be connected itself.

Graph Size	Side Length	Nodes	Edges	Steiner Sites	Terminals
I	4	64	384	16	32
II	6	216	1296	54	108
III	8	512	3072	128	256
IV	10	1000	6000	250	500
V	12	1728	10368	432	864

Table 10.2.: Graph sizes for 3D torus instances.

10.5. IGen

We have selected the graph sizes as listed in Table 10.3. Edge costs are defined by the metric distance between nodes. However, inter-AS edges are weighted three times the original (metric) cost. All PoPs are selected to be Steiner sites and terminals are distributed uniformly at random. Figure 10.2 depicts an example instance.

The costs for installing processing functionality on Steiner sites again depends on the diameter of the graph. We use again 0.125, 0.25, 0.5, 1, 2 times the diameter of the graph as *expected* cost. In contrast to the data center scenarios, we can argue that the costs for activating Steiner sites depends to some extent on the location. We therefore add some noise. If c is the expected cost, then Steiner site costs are distributed according to the following distribution $c + \mathcal{U}(-c/20, c/20)$, such that the cost may vary by at most 10%.

Lastly, as inter-AS links represent backbone connections, we add a fixed capacity of three on top of the randomly selected capacity that ranges between one and three.

Graph Size	Nodes	Intra-AS Edges	Inter-AS Edges	Steiner sites	Terminals
I	800	3040	294	79	160
II	1600	6032	678	163	320
III	2400	9092	958	233	480
IV	3200	12340	1180	313	640
V	4000	15396	1528	401	800

Table 10.3.: Graph sizes for IGen instaces.

11. Separation & Branching Parameters

In this first part of our computational evaluation we investigate the impact of separation parameters and branching rules on the performance of our branch-and-cut based VirtuCast algorithm. The parameters considered will be discussed shortly below.

11.1. Considered Parameters

creep-flow As discussed in Section 5.4.1, creep-flow is a technique to empirically improve the quality of found cuts.

nested cuts As discussed in Section 5.4.1, employing nested cuts allows to generate multiple cuts in each separation round.

Separation of Terminal Connectivity Inequalities IP-3⋆ In Lemma 5.2 we have shown that including the connectivity inequalities IP-3⋆ can strengthen the formulation. However, as the number of terminals exceeds the number of Steiner sites in most cases, separating these cuts necessitates a large number of additional maximum flow computations.

Scale Parameter We employ an integral maximum flow algorithm to separate connectivity inequalities to circumvent numerical instabilities. Therefore, all real-valued flow values need to be scaled by a certain parameter. While a small scaling factor reduces the runtime of the maximum flow algorithm, it worsens the integral approximation of the flow, such that potentially less cuts are found.

Forcing Cuts into the LP SCIP implements a set of decision rules to decide which of the found cuts shall be included into the LP. The reason for not including all the found cuts is that with each cut, the number of rows in the LP increases and so does the time for solving the linear relaxation. However, SCIP allows to force cuts into the LP, such that all found cuts are included. Forcing the cuts into the linear relaxation might be beneficial, since found cuts can only strengthen the formulation.

Storing Cuts in the Cut Pool SCIP allows to store found cuts in a global cut pool. After having computed the linear relaxation, SCIP can search for cuts in the global cut pool to determine whether the found relaxation is feasible or not. If the relaxation *violates* one or multiple

cuts, these violated cuts are added to the LP and the LP is resolved. As searching the global cut pool for violated cuts is done before calling separation procedures, the computationally expensive maximum flow computations can potentially be avoided.

Branching rules SCIP implements more than half a dozen branching rules. We have selected the two most common ones to test, namely reliable pseudo cost and pseudo cost branching [Ach07]. Both branching schemes build upon the estimation of pseudo costs, estimating the (global) impact on the objective if the variable is fixed to its lower or its upper bound. To obtain initial estimates on the pseudo costs, the reliable pseudo cost branching performs strong-branching at the root [Ach07] In contrast, the (non-reliable) pseudo cost branching rule does not perform strong-branching and therefore only obtains reasonable estimates on the fly after several branchings. Furthermore, we will use the GreedyBranch rule introduced in Section 5.4.2.

11.2. General Methodoloy

Having selected a certain subset of the above parameters, all possible combination of parameters will be tested. To evaluate the impact of a certain parameter, we generally compare the benefits of enabling a feature vs. disabling it on a per instance basis. Considering e.g. the impact of the creep-flow parameter, the different experiments in which creep-flow was enabled are compared to the experiments in which it was disabled. Importantly, this is done in such a way that only executions are compared in which *all the other* parameters agree, such that the impact of parameter is clearly distilled. It must be noted that not all parameters discussed in the above section are independent of one anoter. For example the scale parameter influences both the runtime of separating connectivity inequalities of Steiner sites and terminals. Nonetheless, by considering the improvement on a per instance basis, features that *generally* improve performance can be filtered out.

To evaluate the runtime impact of the parameter settings, we consider the overall separation runtime, the separation runtime per node and the separation runtime at the root. To evaluate the impact on the quality of parameter settings, we consider the dual bound at the root as well as the final dual bound that was obtained. Based on the per execution comparison and as certain features may have both beneficial as well as negative effects, all results are presented as ratios in per cent. Therefore, a value of 10% means

Due to the number of parameters evaluated, experiments are not run on each of the 15 instances (three edge capacity and 5 Steiner cost distributions) per graph size and topology, but only on one. The chosen instance is always the one with medium edge capacities and medium Steiner costs.

11.3. Initial Parameter Validation

Ou initial experiments consider only separation related parameters, namely: creep, nested cuts, separating terminal inequalities, scale, storing cuts and forcing cuts into the LP.

With respect to the scale parameter, settings of $\{100, 1000, 10000\}$ are considered. If nested cuts are enabled, we abort the generation of further cuts at a depth of 5, such that maximally 5 cuts are introduced. With respect to separating terminal inequalities, these are either separated at each node or disabled globally.

Based on the above selection of parameters, 96 different parameter combinations are obtained. As experiments are performed on all topologies and all graph sizes, this yields 1440 experiments overall and we limit the runtime to 30 minutes.

For the experiments, the following common parameters have been used. None of our LP-based heuristics is included, while SCIP's heuristics are called according to their default settings. The global cut pool is assigned a size of 20,000 and cuts are separated at each node. The optimality switch of SCIP is used, such that cut generatos are called more frequently. As SCIP's default settings are used for all other parameters, the reliable pseudo cost branching is used throughout all experiments.

11.3.1. Overview of Results

In Figure 11.1 a global overview of the runtime, the objective gap, the progress of the dual bound (i.e. the final dual bound compared to the root's dual bound), the number of cuts and the number of nodes is given. As one can see, all but the smallest fat tree and the smallest torus instances cannot be solved within 30 minutes to optimality. For the two largest graph sizes, only seldomly solutions are found. The number of cuts decreases when the graph sizes are increased and the same applies to the number of nodes which for the largest graph sizes are generally below 50. Most importantly however, the improvement of the dual bound with respect to the root's dual bound is very limited.

To explain this behavior, the runtime allocation of the most time consuming parts of the solver are depicted in Figure 11.2. As one can observe, (especially) diving heuristics are running for up to 1000 seconds, and therefore may consume more than half of the overall runtime, which is limited to 1800 seconds. Together with the (per default) enabled reliable pseudo cost branching, which approximately uses 400 seconds on the larger instances and LP solving times of multiple hundred seconds, this leaves only little time for exploring nodes and separating inequalities.

Since the runtime of the final solver will be limited by one hour instead of 30 minutes, and as on some instances less than 5 seconds is spent in separation procedures, the obtained results are unreliable and do not allow to project the impact of the parameters under investigation on a runtime of one hour. Therefore, another set of experiments with a runtime of one hour was undertaken and is reported on in Section 11.4, where diving heuristics are disabled and branching rules are investigated, too. To reduce the number of overall parameter combinations for this second parameter validation, we nonetheless evaluate the impact of some parameters in the next secion.

11.3.2. Detailed Results of Parameter Validation I

Figures 11.3 - 11.5 present the detailed analysis of the impact of the following parameters: force cuts into the LP (see Figure 11.3), store cuts in the global cut pool (see Figure 11.4),

separate terminal connectivity inequalities (see Figure 11.5). In all of these figures, the performance of when a feature is enabled is compared to the performance when it is disabled.

Forcing Cuts into the LP

According to Figure 11.3 forcing all found cuts into the LP has a very negative impact on IGen instances, lowering the final dual bound by 12% on graph size III. Note that this is despite the fact that the root relaxation time is considerably reduced on between 5% and 10% the time necessary when cuts are not forced into the LP. This parameter has no impact on the final dual bound of 3D torus and fat tree instances. As the solution time of the linear relaxations increases as all found are cuts are used, we will disable forcing cuts into the LP and rather rely on the techniques implemented in SCIP to determine the *good* cuts to use [Ach07].

Storing Cuts in the Global Cut Pool

In Figure 11.4 the impact of storing cuts in the global cut pool is presented. Enabling this feature has seemingly no impact at all, as all final dual bounds are essentially the same and neither the overall separation time is not decreased. As storing the found cuts and separating introduces additional computational overhead and does not improve the performance, it will generally disabled.

Separation of Terminal Connectivity Inequalities

Considering the separation of terminal connectivity inequalities (see Figure 11.5 we notice that the time spent in separation procedures per node as well as overall only increases slightly. This is a rather surprising result, as separating terminal inequalities at each node requires at least one additional maximum flow computation per terminal.

Since the quality of the dual bound is not influenced to any extent, in the next parameter validation (see Section 11.4) the frequency of separating terminal inequalities will be lowered to try to improve the performance / runtime trade-off.

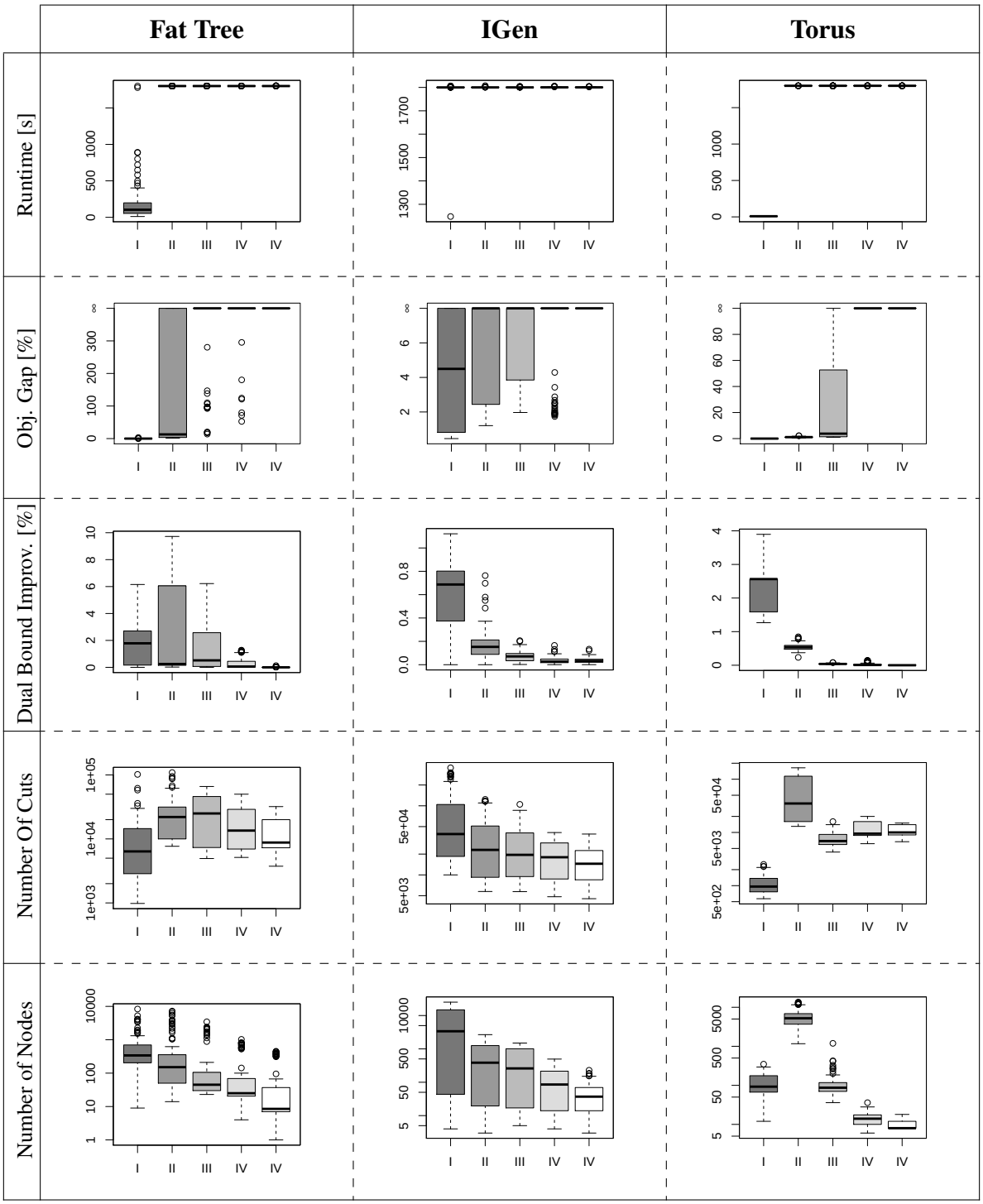


Figure 11.1.: Overview of the general performance of the solver in the initial separation evaluation for the different topologies and the different graph sizes.

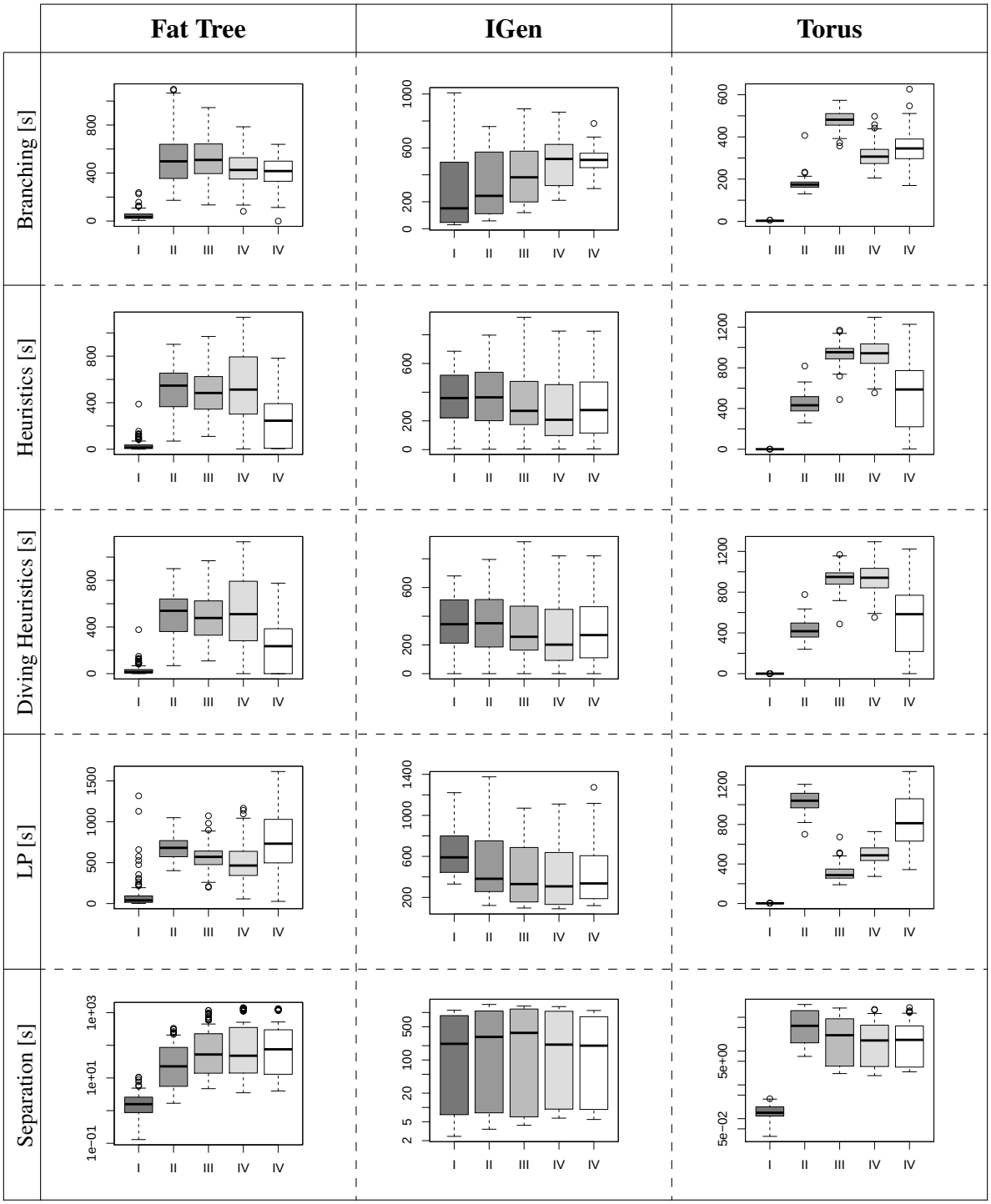


Figure 11.2.: Runtime analysis of the solver during the initial parameter validation. The branching runtime contains the time for solving linear relaxations when strong branching is performed. The heuristic runtime analogously contains the runtime for solving linear relaxations when e.g. diving is performed. The LP runtime itself therefore only denotes the runtime spent for solving the relaxations of nodes of the branch-and-bound tree. The separation runtime contains the maximal flow computations without resolving the linear relaxations.

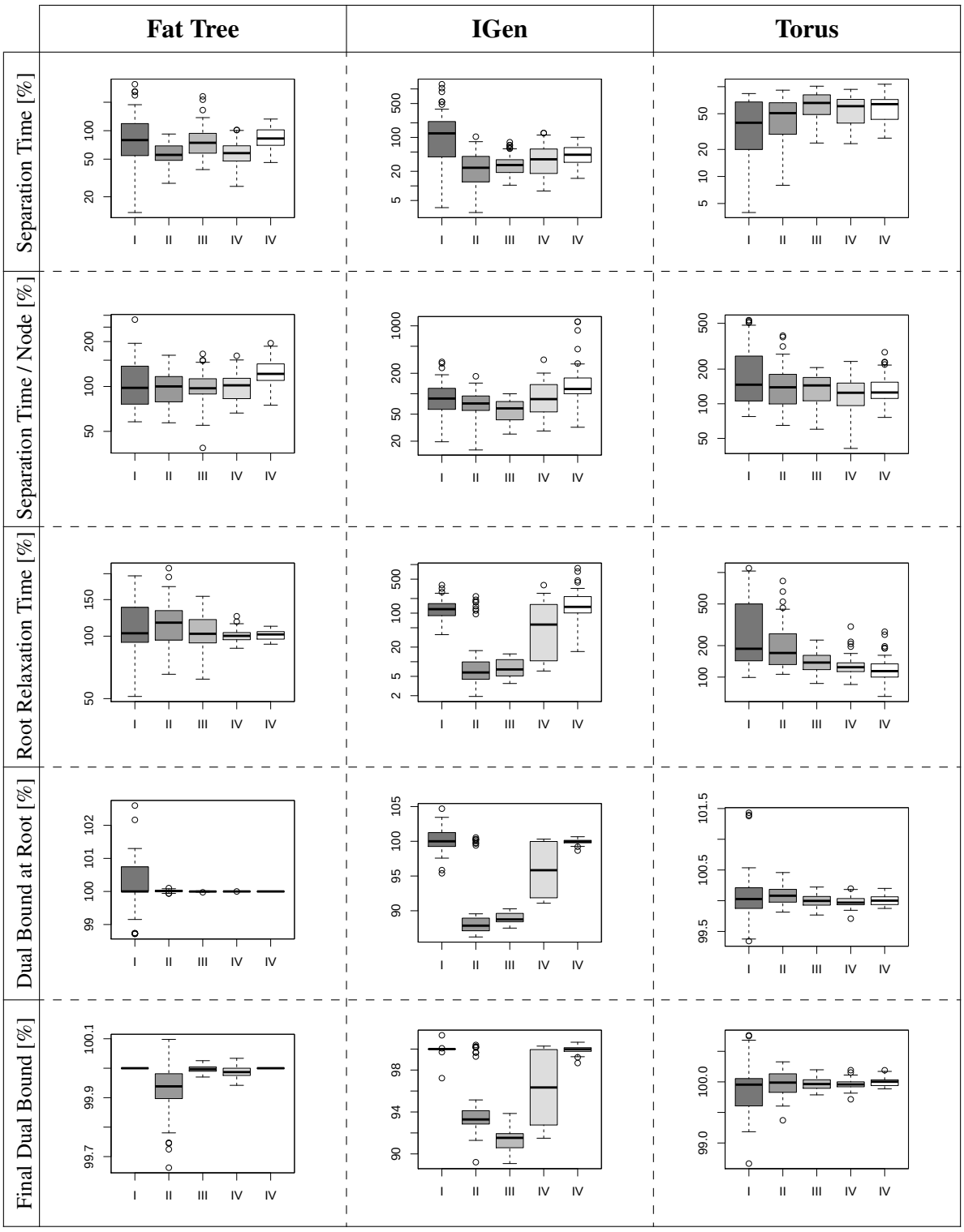


Figure 11.3.: Impact of force adding cuts into the LP for all graph sizes on all topologies.

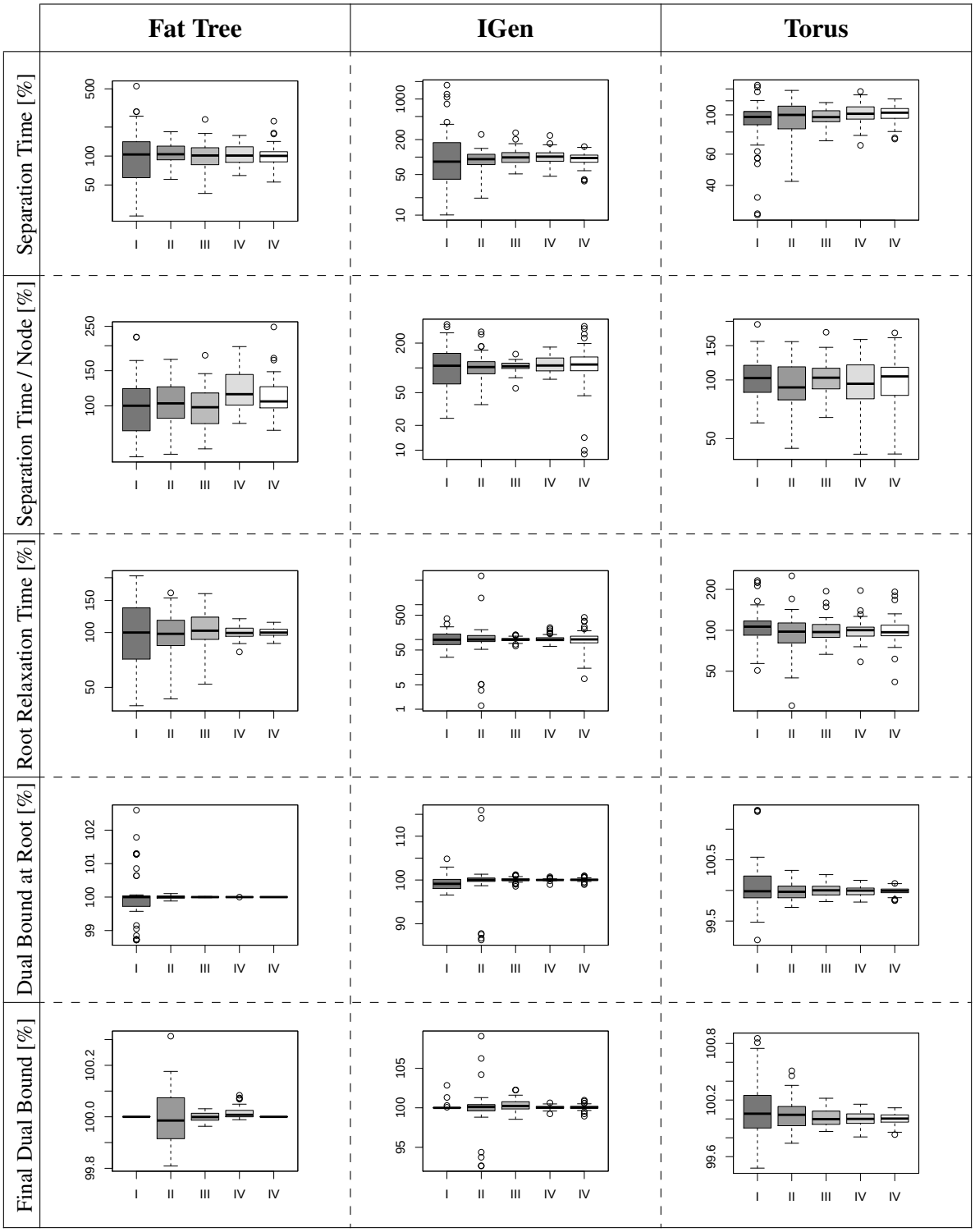


Figure 11.4.: Impact of storing cuts in the global cut pool for all graph sizes on all topologies.

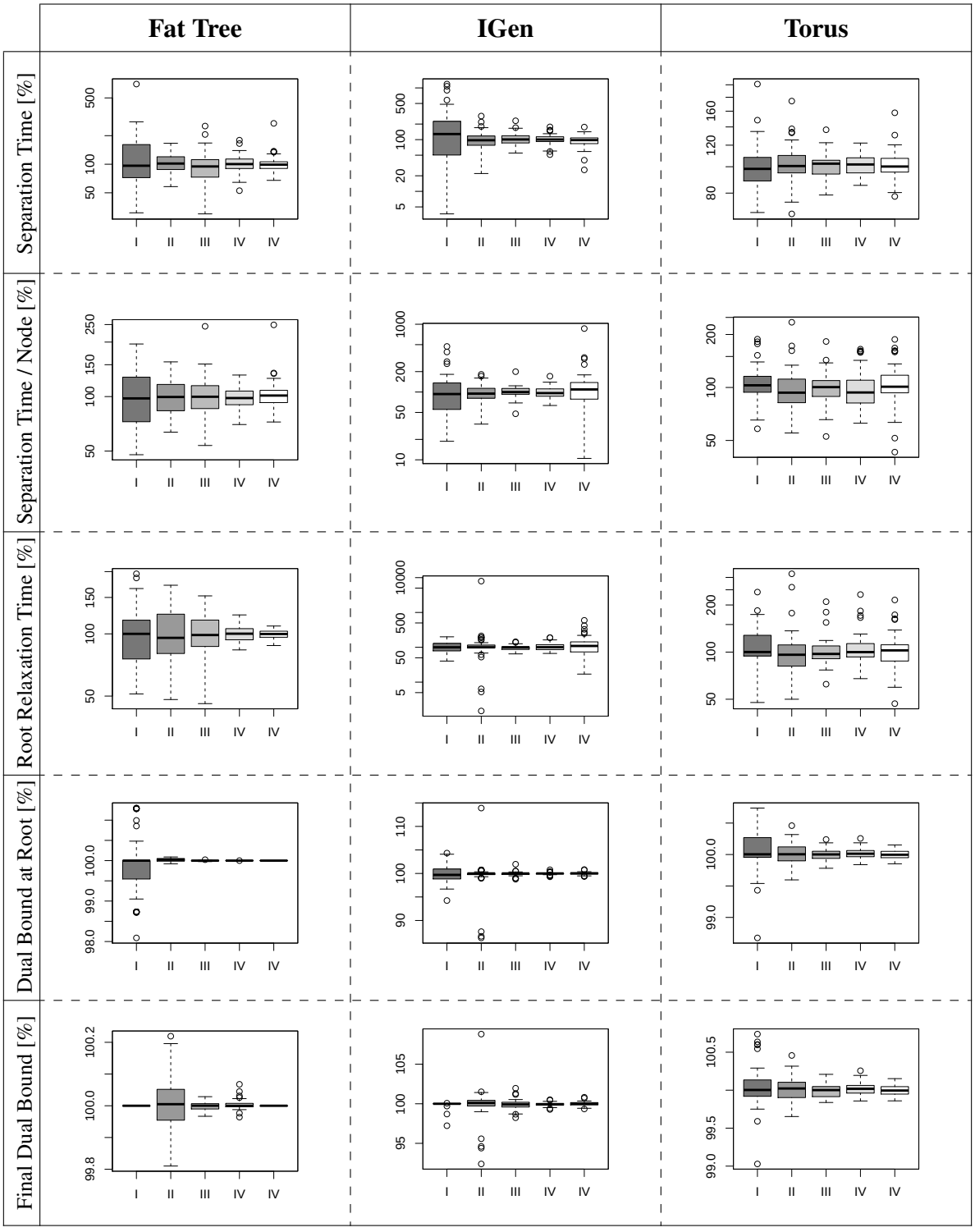


Figure 11.5.: Impact of seperating terminal connectivity inequalities for all graph sizes on all topologies.

11.4. Final Parameter Validation

As discussed in Section 11.3.1, diving heuristics as well as the per default enabled reliable pseudo cost branching rule consumed a majority of the runtime, which was limited to 1800 seconds. As diving heuristics failed to produce solutions on most of the larger instances, these are disabled in the final parameter validation. Furthermore, as the reliable pseudo cost branching may consume up to 600 seconds, the study presented henceforth will also consider different alternative branching rules. To yield reliable results, the runtime limit is increased to the runtime limit of the final experiments, namely one hour.

The results of Section 11.3.2 suggest that forcing cuts into the LP and storing cuts in the global cut pool have no (positive) effect. To reduce the number of experiments, these parameters are not considered. The number of experiments is further reduced by only considering scale factors of $\{1000, 10000\}$, as Koch et al. advise a setting of 100,000 [KM98].

The final parameter validation experiments therefore consider the following parameters: creep, nested cuts, separating terminal inequalities, scale and branching rules. The branching rules employed are reliable pseudo cost, pseudo cost and the greedy branching scheme presented in Section 5.4.2. As discussed in Section 11.3.2, the separation of terminal connectivity inequalities does not worsen the final lower bound, even though the time spent in separation procedures increases vastly. Therefore, in the final set of experiments, we consider separating them with a frequency of 10 versus only separating them at the root. A frequency of 10 means that the inequalities are separated at all nodes of the branch-and-bound tree whose depth is a multiple of 10.

Similarly to the inital parameter validation, for each topology and graph size the medium edge capacitiy and medium Steiner cost instance is used. Based on the above selection of parameters, 48 different parameter combinations are tested on each of the 15 topologies and graph sizes, yielding 720 experiments overall.

The evaluation of this second set of experiments follows the same outline as the initial parameter validation. In Section 11.4.1 first the general performance of the solver will be discussed and the reliability of the results established. In Section 11.4.2 the impact of the parameters are discussed in-depth. Based on this discussion, the final set of parameters for all further experiments is summarized in Section 11.5.

11.4.1. Overview of Results

The performance overview of the solver is presented in Figure 11.6. Note that even though the diving heuristics have been disabled, the runtime of one hour allows to solve *more* instances than in the initial set of experiments. Importantly, the improvement of the dual bound with respect to the root relaxation has been slightly decreased across all topologies and graph sizes (cf. Figure 11.1). The authors believe this to be due to the lack of high quality solutions, which were obtained previously by the diving heuristics.

Considering the runtime allocation of the different components of the solver (see Figure 11.7), a much larger fraction of time is spent in the separation procedures of connectivity inequalities. Furthermore, the median of the runtime of heuristics lies around 200 seconds and

the time spent in branching is only above a few seconds if reliable pseudo cost branching is applied.

11.4.2. Detailed Results of Parameter Validation II

Figures 11.8 - 11.11 present the detailed analysis of the impact of the parameters creep-flow, nested cuts, separation of terminal inequalities and scale.

For evaluating the impact of branching rules, Figures 11.12 - 11.14 compare the results of the three testes branching rules on 3D torus, IGen and fat tree instances respectively.

Creep-Flow

The runtime and qualitative impact of creep-flow is depicted in Figure 11.8. While the quality of the final dual bound increases by 1% to 3% on IGen and 3D torus instances, the impact on fat tree instances is negligible. Importantly, the dual bound is increased even though the time spent in separation procedures is increased by a factor of up to 50. This increase in runtime is due to the fact that edges not used in the relaxation's solution carry flow, thereby increasing connectivity, such that the corresponding flow computations require more time.

This shows that by employing creep-flow the quality found cuts is substantially improved. Thus, creep-flow will be enabled in all further experiments.

Nested Cuts

In contrast to creep-flow, employing nested cuts to generate multiple cuts for each Steiner site or terminal does generally not improve the dual bound while increasing the separation time per node (see Figure 11.9). Nested cuts will therefore be disabled for all further experiments.

Terminal Connectivity Inequalities

Separating terminal connectivity inequalities at depth levels $\{0, 10, 20, \dots\}$ compared to separating them only at the root (depth level 0), does neither improve the final dual bound nor increase the separation time significantly. Again, this is highly interesting as the number of terminals is generally larger than the number of Steiner sites. Even though no distinct qualitative advantage is obtained, terminal connectivity inequalities will be separated with a frequency of 10 for all further experiments, as they have the potential to strengthen the formulation and we only considered one of the 15 Steiner cost and edge capacity combinations.

Scale

Considering the scale parameter, similarly no distinct qualitative improvement is obtained when using a scale of 10,000 over a scale of 1,000 (see Figure 11.11). Furthermore, the increase in the separation time lies between a factor of 1 to 2. As choosing a smaller scale parameter decreases the worst-case runtime of the emplyoed maximum flow algorithm, the scale of 1,000 will be used for all further experiments.

Branching Rules

The impact of the different branching rules is depicted in Figures 11.12-11.14 for the 3D torus, IGen and fat tree topologies. First note that the dual bound at the root is independent of the branching rule employed, whereas the initial probings when strong-branching is applied in the reliable pseudo cost branching rule may increase the root relaxation time by a factor of of up to 5.

As the runtime of the (non-reliable) pseudo cost and the greedy branching rules are bounded by a few seconds, the remaining time for solving linear relaxations and separating connectivity inequalities remains the same. However, when reliable pseudo cost branching is employed, the time spent in separation procedures may be reduced by a factor of 50% on 3D torus instances.

Considering the quality of the final dual bound, the following hierarchy can be established. Both the reliable and non-reliable pseudo cost branching perform better than the greedy branching strategy and reliable pseudo cost branching (slightly) outperforms non-reliable pseudo cost branching on smaller graph sizes. As the advantage of using the reliable variant diminishes on larger instances while the number of nodes considered is reduced, the reliable variant will be employed for graph sizes I and II, while the non-reliable variant is employed for graph sizes III,IV and V.

11.5. Final Separation & Branching Parameters

In the following, the final separation and branching parameters employed for all further experiments (unless otherwise noted) are summarized.

Common to all graph sizes, creep-flow is enabled, nested cuts are not used, terminal connectivity inequalities are separated with a frequency of 10 and the scale is set to 1,000.

For graph sizes I and II the reliable pseudo cost branching rule is used while for graph sizes III, IV and V the non-reliable variant is employed.

PERFORMANCE OVERVIEW OF PARAMETER VALIDATION II

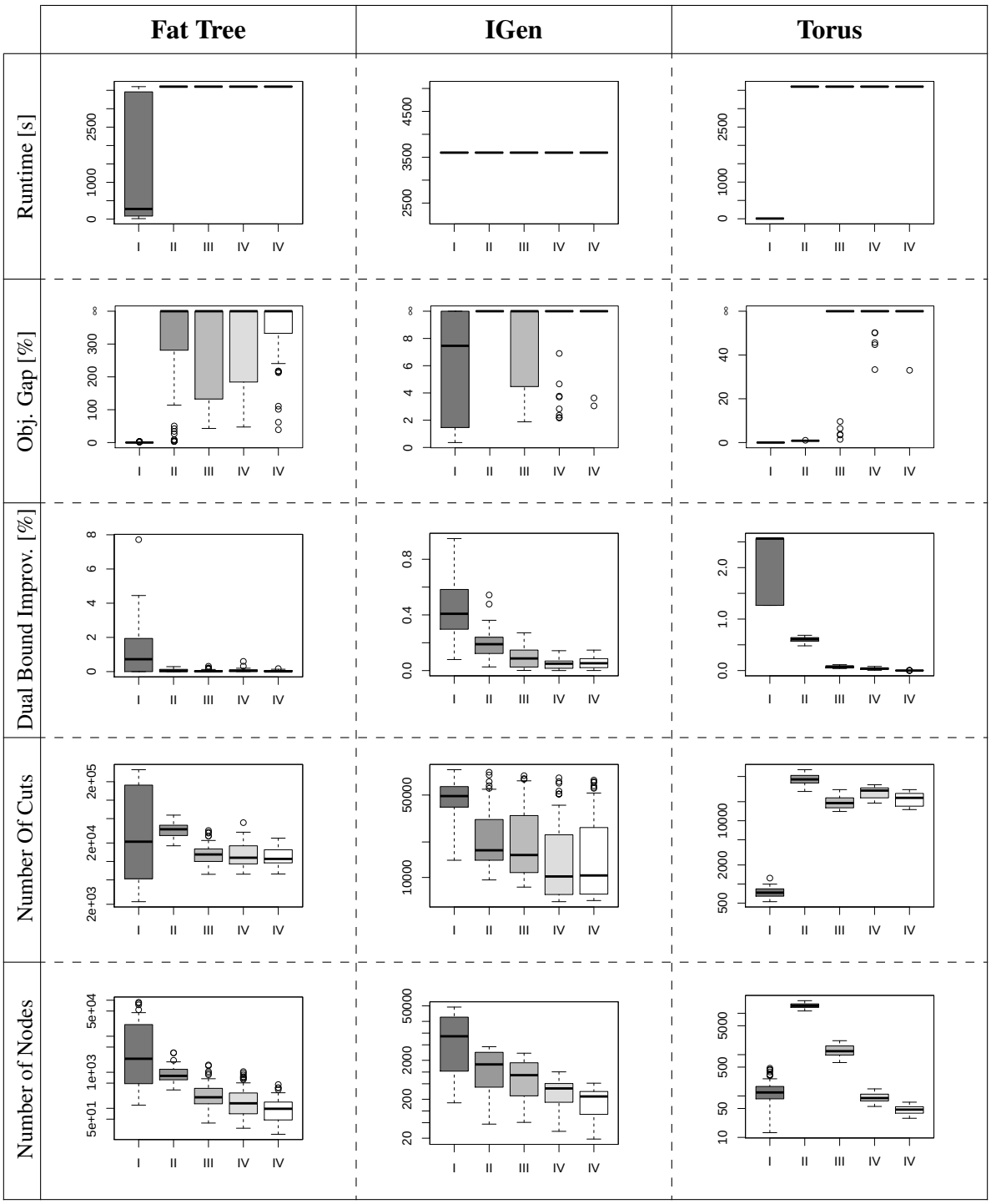


Figure 11.6.: Overview of the general performance of the solver in the final parameter validation for the different topologies and the different graph sizes.

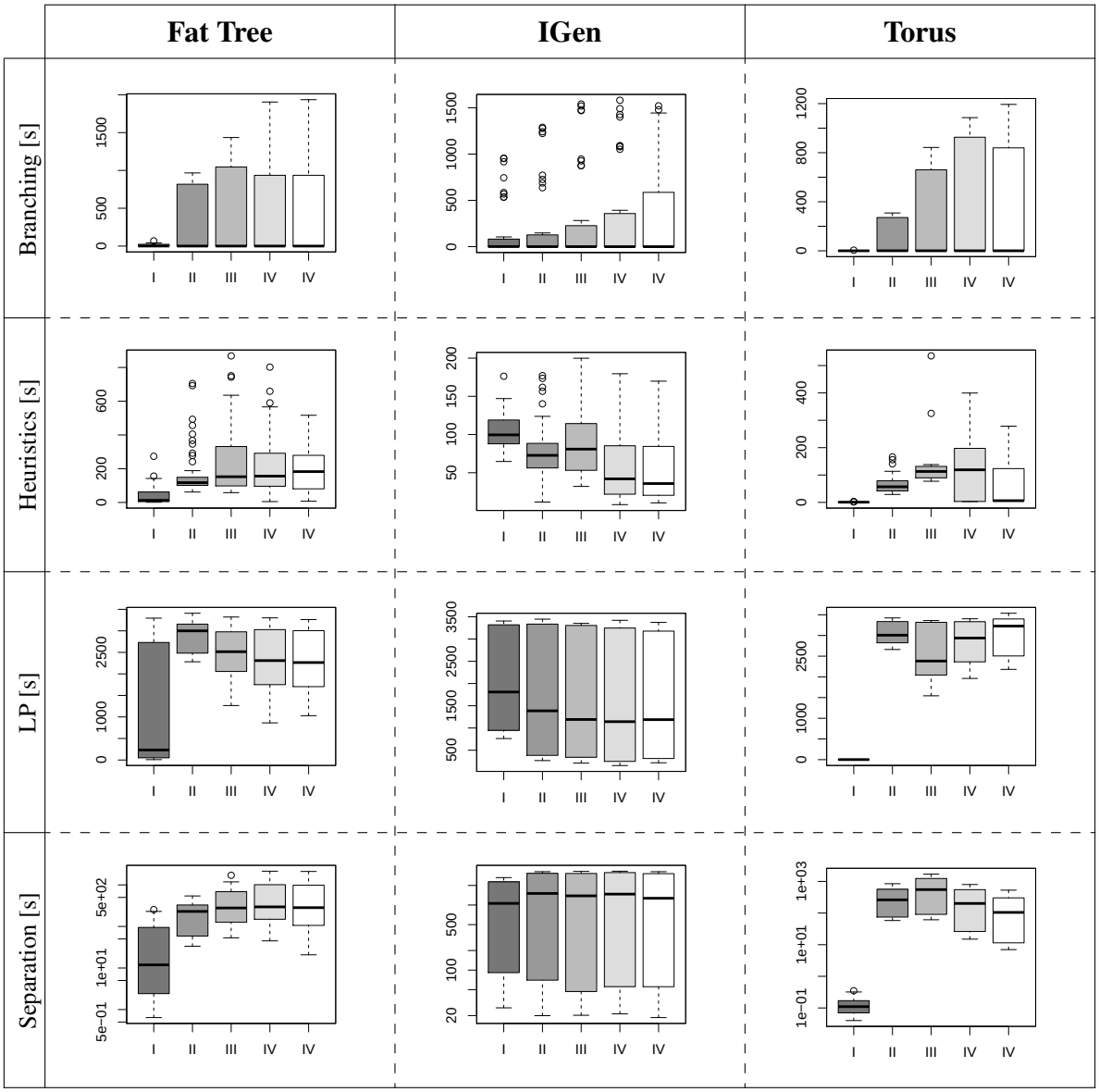


Figure 11.7.: Runtime analysis of the solver during the final parameter validation. The branching runtime contains the time for solving linear relaxations when strong branching is performed. The LP runtime itself therefore only denotes the runtime spent for solving the relaxations of nodes of the branch-and-bound tree. The separation runtime contains the maximal flow computations without resolving the linear relaxations.

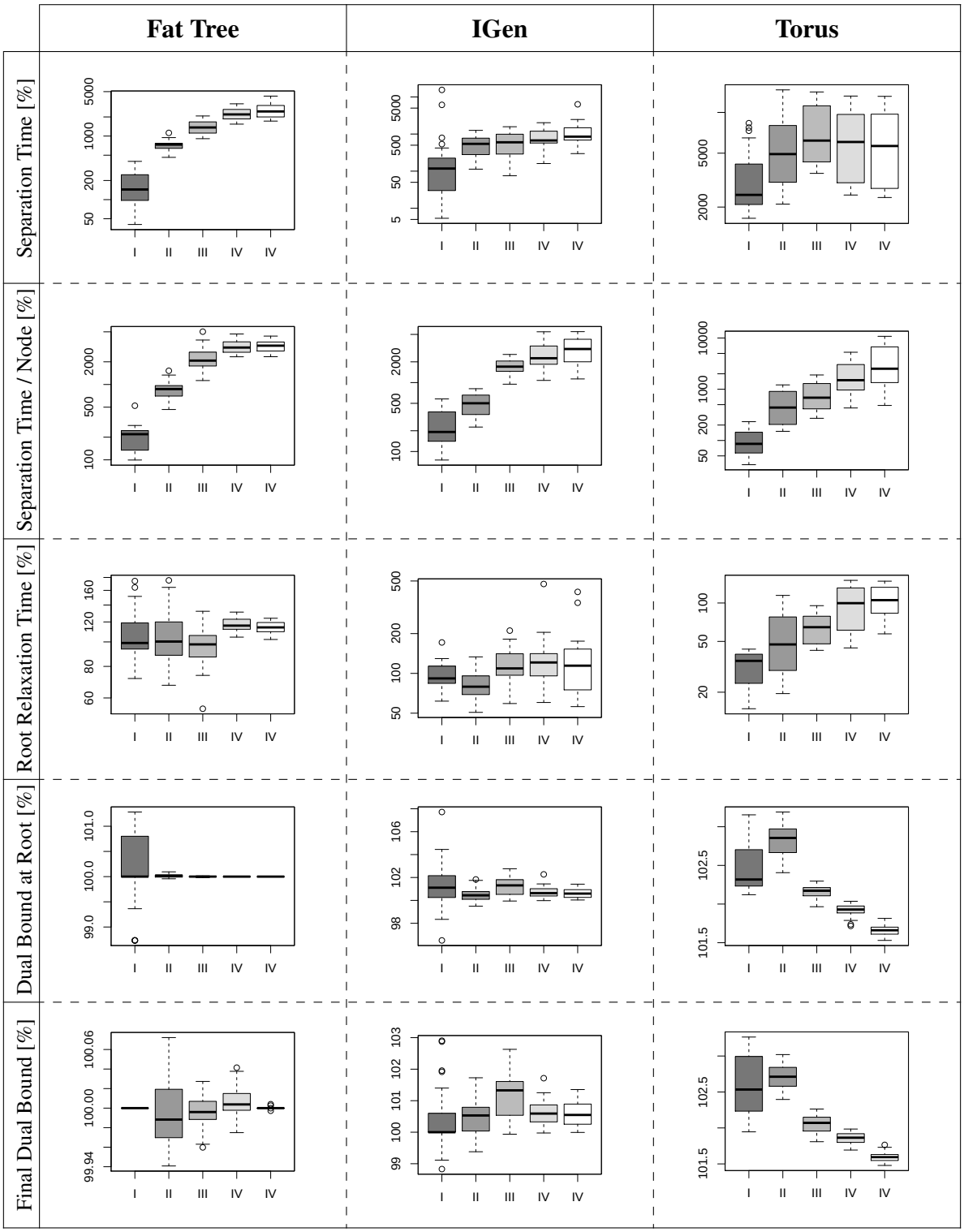


Figure 11.8.: Impact of enabling creep-flow for all graph sizes on all topologies.

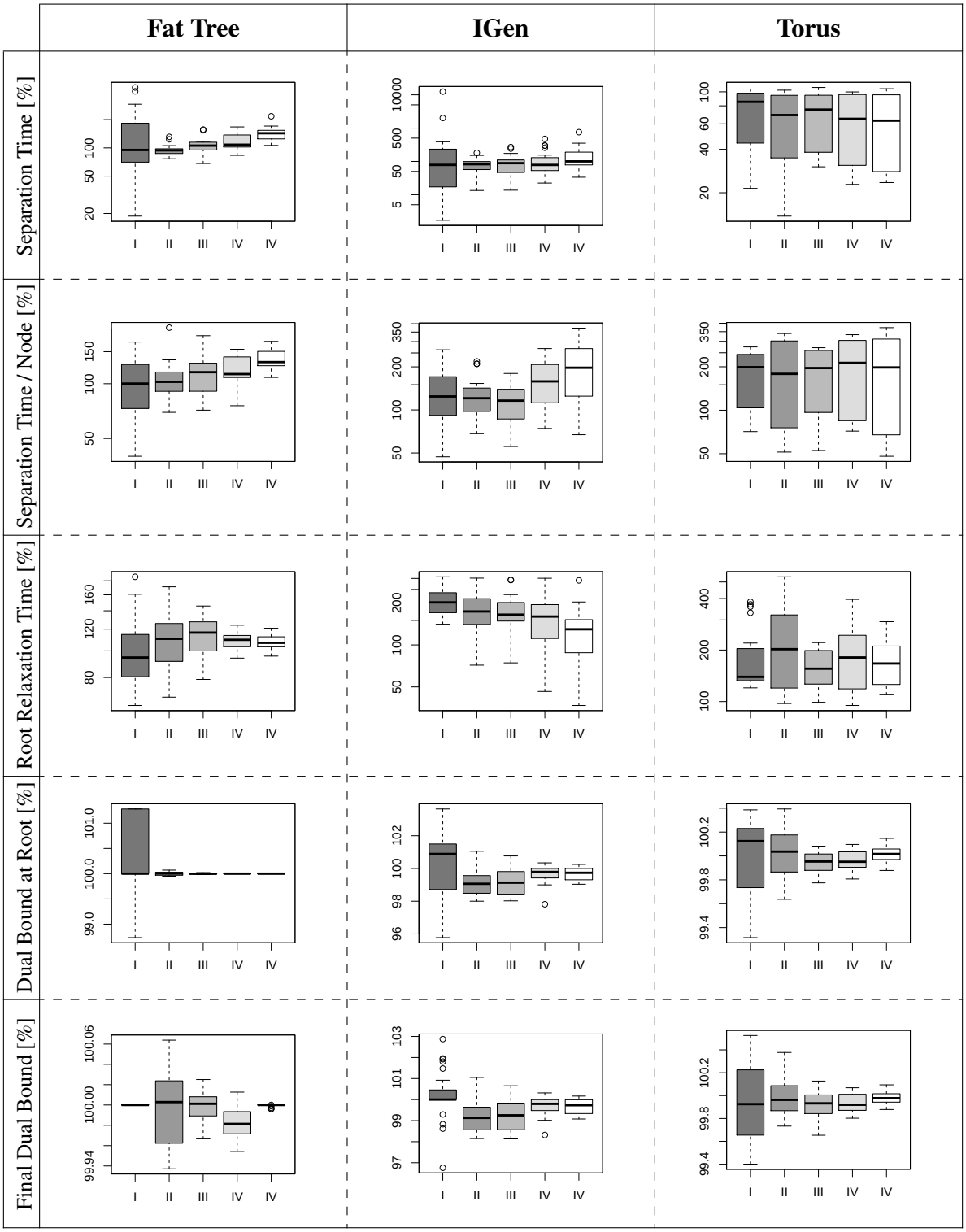


Figure 11.9.: Impact of using nested cuts with a maximal depth of 5 for all graph sizes on all topologies.

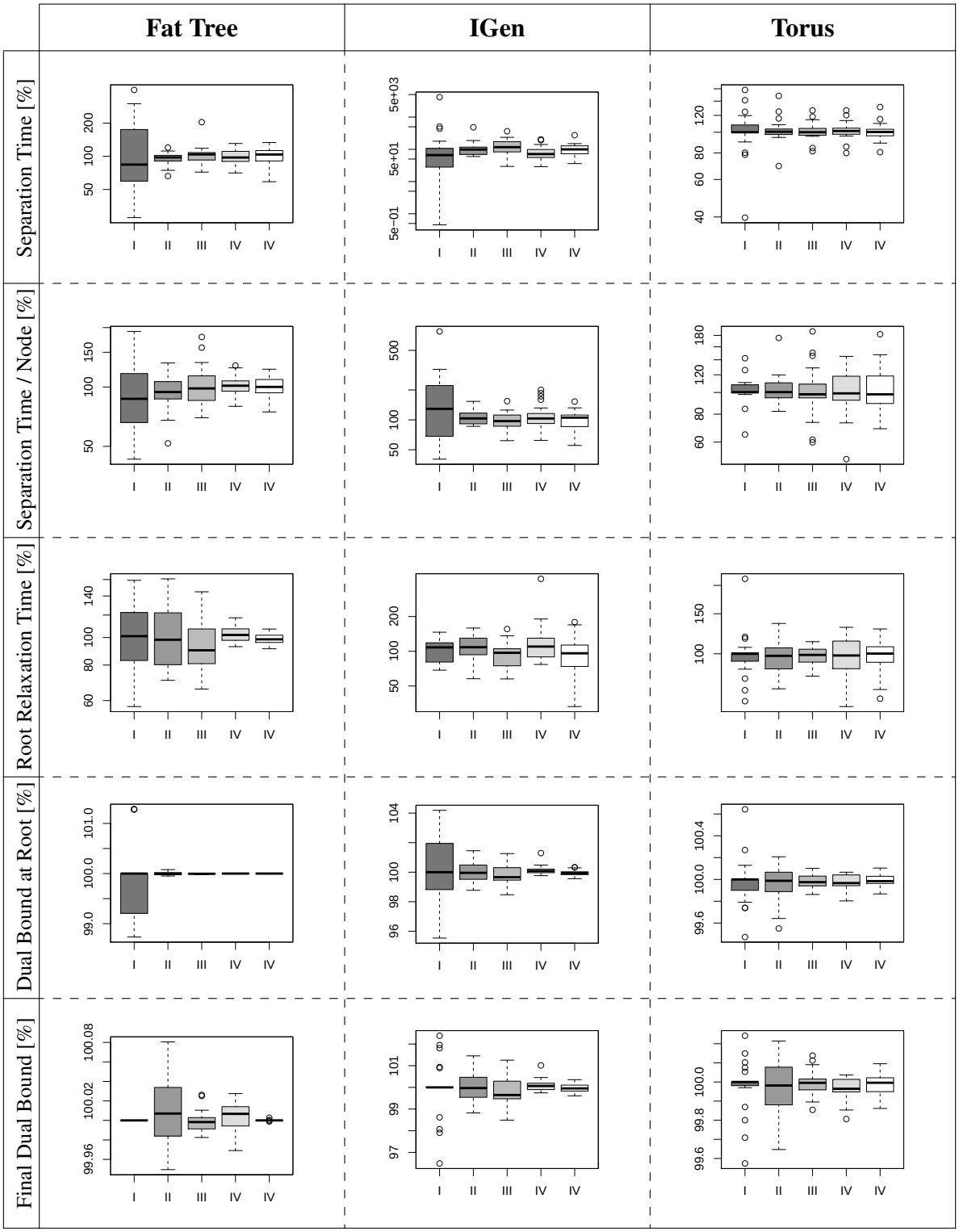


Figure 11.10.: Impact of separating terminal connectivity inequalities with a frequency of 10 instead of separating them only at the rootfor all graph sizes on all topologies.

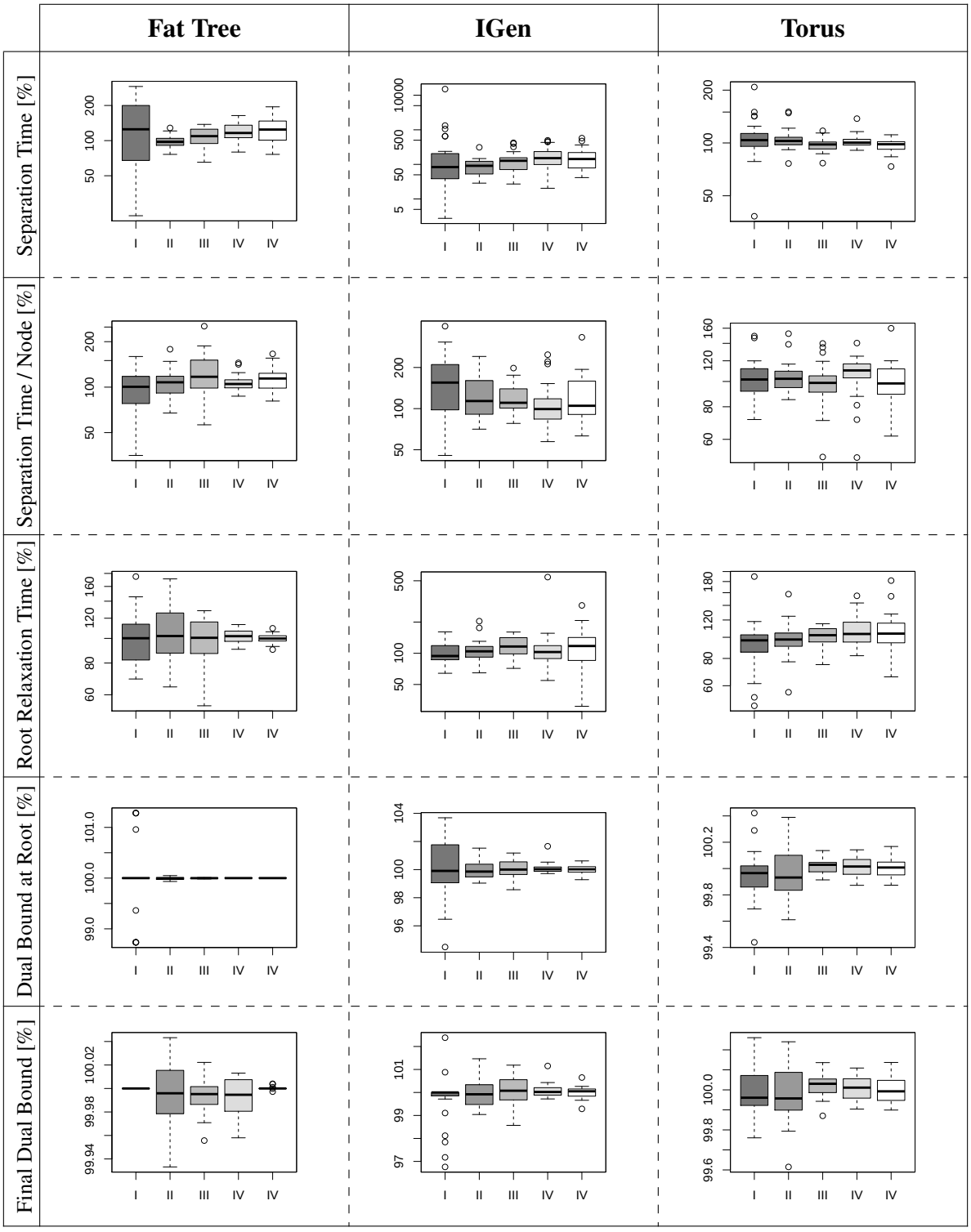


Figure 11.11.: Impact of using a scale of 10,000 instead of 1,000 for all graph sizes on all topologies.

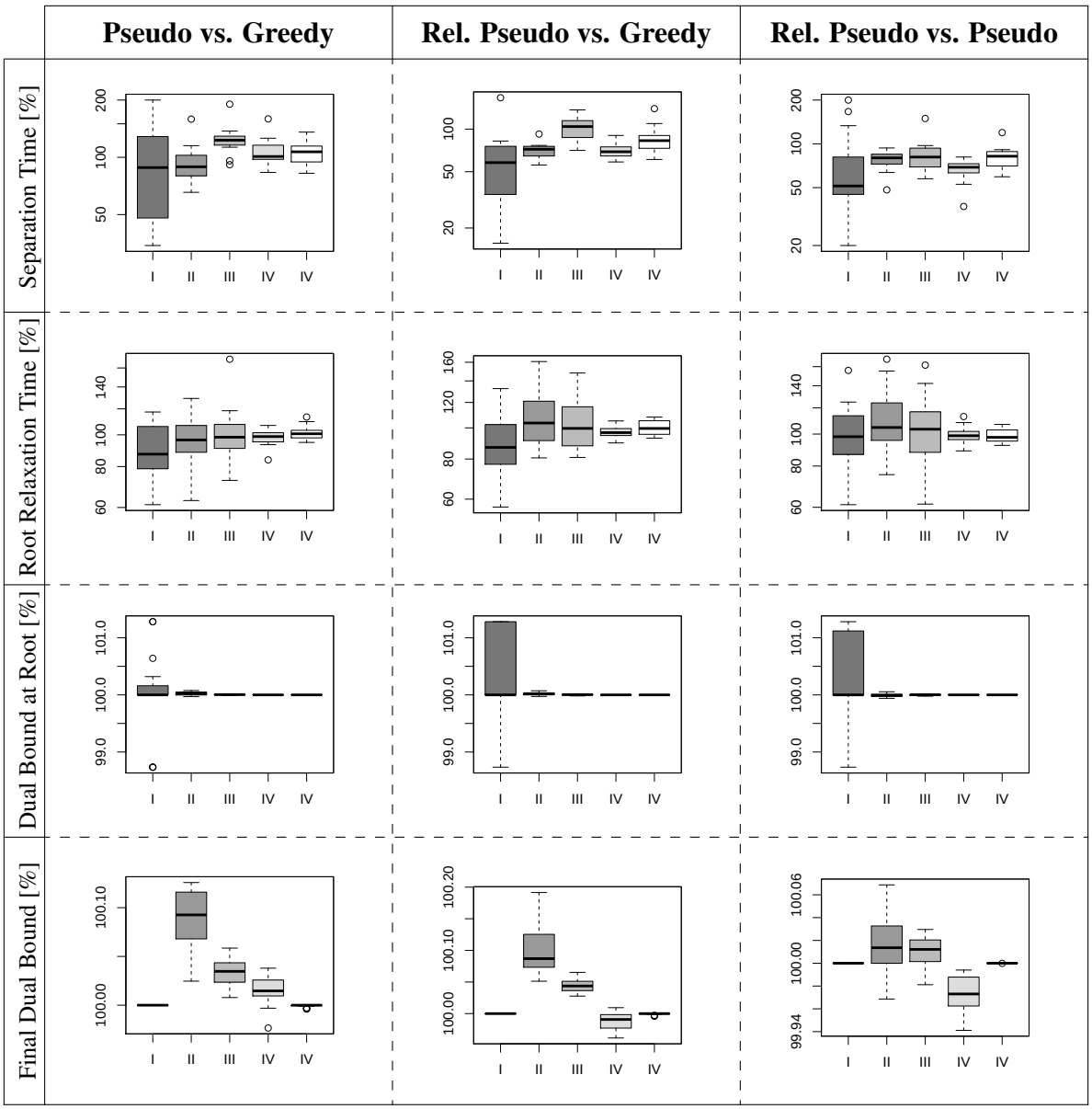


Figure 11.12.: Impact of branching rules on instances of the 3D torus topology. The results of the first mentioned branching rule are compared to the results of the second mentioned branching rule.

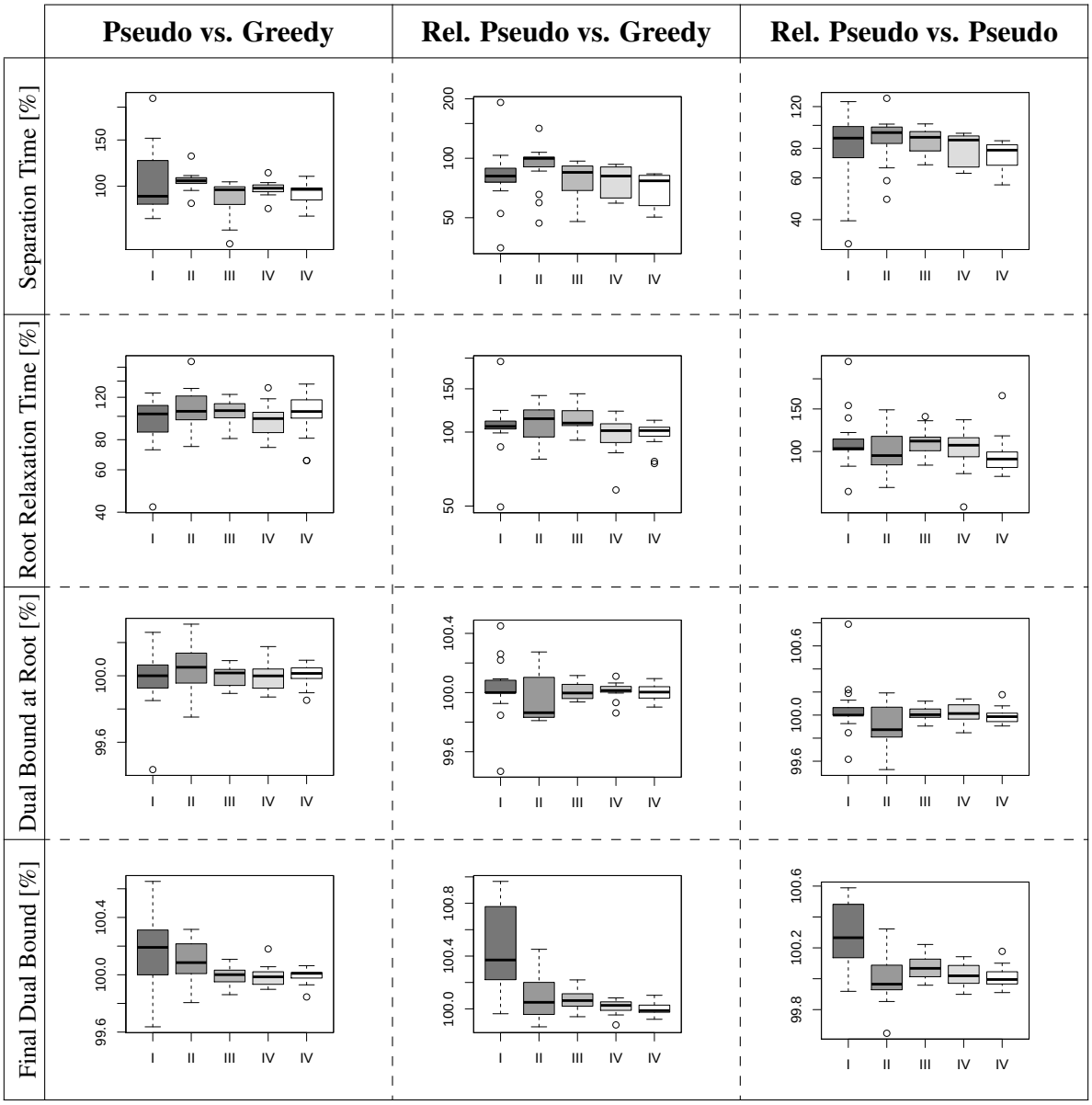


Figure 11.13.: Impact of branching rules on instances of the IGen topology. The results of the first mentioned branching rule are compared to the results of the second mentioned branching rule.

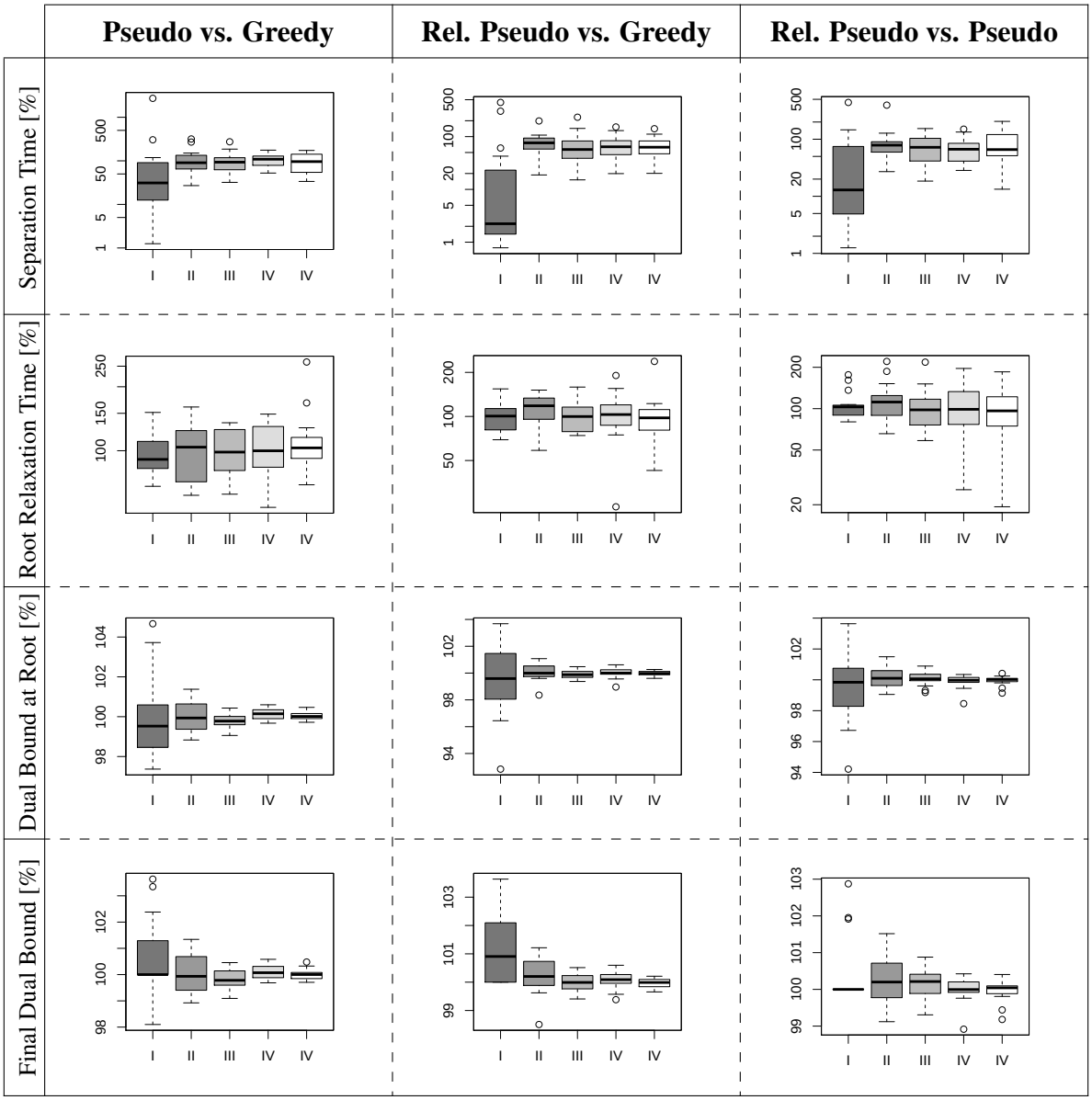


Figure 11.14.: Impact of branching rules on instances of the fat tree topology. The results of the first mentioned branching rule are compared to the results of the second mentioned branching rule.

12. Performance of LP-Based Heuristics

In this section the results of our initial LP-based heuristics' performance assessment are presented. Instead of considering only one of fifteen instances per graph size, all combinations of Steiner cost and edge capacity distributions are considered. The goal of this section is to investigate the performance of the LP-based heuristics that were introduced in Section 8 in terms of runtime and solution quality and to establish both topology-independent and topology-dependent observations regarding the trade-off between runtime and solution quality.

Sections 12.1 and 12.2 introduce the methodology and the computational setup respectively. In Section 12.3 a global overview of the results is given which is then followed in Section 12.4 by a detailed discussion of the performance of each heuristic on each topology. This discussion will yield a (topology-dependent) selection of heuristics to include in the final VirtuCast solver. Table 12.1 lists the LP-based heuristics under discussion and introduces abbreviations used throughout this section.

12.1. Methodology

The comparative investigation of LP-based heuristics within a branch-and-bound framework is delicate, as the heuristics are guided by the employed branching and node selection rules (see [Ach07]) and the heuristics' results theirselves influence the branching by potentially cutting off subtrees in the bounding step (cf. introduction of Section 6). The reliability of the results therefore crucially depends on the employed methodology. As a consequence, our experimental setup is guided by the following principles.

1. All heuristics are always called at the same nodes, such that the performance *with respect to a single given node* can be compared.

2. The order in which heuristics are called *does not influence* the result of the execution of any other heuristic.

3. The execution of heuristics should only influence the solver's state to a minimal amount, such that the solver *does not implicitly learn* information from the execution of heuristics.

To comply with the first two points we have implemented a *master heuristic* within the SCIP framework, that delegates the call sequentially to all LP-based heuristics. Only after having obtained the result from all heuristics, the best solution found is returned to SCIP, such that e.g. the primal bound is only updated after all heuristics have been executed.

FDR	Heuristic FlowDecoRound	(see Algorithm 8.1)
GD	Heuristic GreedyDiving,	(see Algorithm 8.4)
GSD	Heuristic GreedySteinerDiving	(see Definition 8.5)
FGSD	Heuristic FastGreedySteinerDiving	(see Definition 8.6)
MS	Heuristic MultipleShots	(see Algorithm 8.5)
MSS	Heuristic MultipleShotsSquared	(see Definition 8.7)

Table 12.1.: Abbreviations for LP-based heuristics.

With respect to the last point, instead of employing the (reliable) pseudo cost branching rules that have been found most effective in Section 11, the GreedyBranch rule introduced in Definition 5.13 is utilized. While this branching rule might be beneficial for some of the heuristics, the author could not reliably determine the influence of the execution of separation procedures on the pseudo costs within the SCIP framework [Ach09]. As the GreedyBranch rule does not rely on pseudo costs and was implemented by the author, such side effects can be ruled out when utilizing the GreedyBranch rule.

Another important methodological decision made is to disable the primal bound during the execution of *all heuristics*. As discussed in Section 8.4, if a heuristic relies on LP diving, then the construction can be aborted prematurely, once the dual bound exceeds the primal bound, in case the objective value of a solution under construction is strictly bounded by the dual bound of the LP relaxation. By disabling the primal bound, the *general quality* and the *general runtime* of the heuristic can be measured, as it would otherwise not be possible to know whether the heuristic would have produced a solution and of which quality it would have been. On a similar note, the Algorithm PruneSteinerNodes is used *for each* heuristic to (try to) improve the objective value, if a solution has been found. While this approach may disagree with the formal definition and the design discussions of the heuristics (see Section 8), it will actually allow to *argue for* the design decisions made.

12.2. Computational Setup

Having discussed the methodology above, we will now describe the exact computational setup used. For each topology and each graph size fifteen instances are considered which are generated according to the Steiner cost and edge capacity distributions described in Section 10.2, yielding 225 instances overall. As six heuristics are employed, the time limit for each experiment is set to two hours.

The master heuristic, which calls all other heuristics, is executed with a frequency of 10, such that it is executed at each node of depth $\{0, 10, 20, \dots\}$. To guarantee that each heuristic is called at least once a time limit of 1000 seconds is enforced for each heuristic, such that the solution construction is immediately aborted once this time runs out. To allow for evaluating whether Algorithm PruneSteinerNodes should be employed, the objective value of the original solution as well as the objective value after pruning are stored. Similarly, the runtime of executing the heuristic itself and exexuting Algorithm PruneSteinerNodes is measured separately.

With the exception of using the GreedyBranch rule, the separation parameters of Section 11.4 are utilized.

12.3. Overview of Results

Figures 12.1 to 12.10 depict the results of the evaluation of the LP-based heuristics. The following measures are used to evaluate the performance: the frequency with which solutions were constructed, the quality of the found solutions and the runtime. Following our methodology, the quality of solutions found before and after Steiner nodes were pruned and the runtimes for executing the heuristic and for pruning Steiner nodes will be discussed separately.

The evaluation will proceed as follows. First a general overview is given

12.3.1. Efficiency in Finding Solutions

Figures 12.1 and 12.2 depict the absolute number of found solutions and calls to the heuristic and their ratio respectively. Considering the frequency with which solutions were found, we note that on fat tree and IGen instances nearly each call yields a feasible solution for all heuristics except FDR. On 3D Torus instances the median of finding feasible solutions still is 100% but more variance is evident. Considering the absolute number of found solutions in Figure 12.1, patterns of unsuccessful calls, represented as black bars, emerge. Even though this might be coincidential, it is reasonable to assume that these instances are either generally hard to find solutions for or that some of the explored nodes, on which *all* the heuristics were called, did not allow for constructing a feasible solution *using only* the Steiner sites that were not a priori disabled in the branching. We note the following observation.

Observation 12.1: The LP-based heuristics are capable of finding solutions reliably across all topologies and graph sizes.

12.3.2. Solution Quality

Figures 12.3 to 12.7 depict the solution quality after pruning Steiner nodes and the respective improvement in the objective gap. Figure 12.3 gives an overview over the objective gap with respect to the final dual bound, showing that for all instances a gap of around 5% can be obtained (if a solution is found). As the main goal is to compare the performance of the heuristics *with each other*, Figure 12.4 depicts the objective gap with respect to the *best primal* solution found per instance. As can be directly observed, the (median) quality of solutions with respect to this measure follows the order in which the heuristics' results are presented *on all instances*. Importantly, this observation agrees with the design criteria of the heuristics, such that GD performs better than GSD which in turn performs better than FGSD. Similarly, by squaring the probabilities to use a Steiner site in the MSS heuristic a (slight) improvement in the solution quality can be observed. Clearly, the FDR heuristic exhibits the worst performance (even after pruning) and we note the following observation.

Observation 12.2: The quality of solutions found generally decreases according to the monotonic order: GD, GSD, FGSD, MSS, MS, FDR.

Considering the fact that the heuristics are executed multiple times at different nodes and that at least MS(S) and FDR are randomized, the question arises whether these heuristics can achieve solutions *close to the best primal solution* or whether there is an inherent qualitative trade-off in using these. Figure 12.6 answers that question by depicting how close *the best solution* generated *by each heuristic* comes to the overall best primal solution found (per instance). With respect to this measure, the qualitative order already established becomes even clearer. While GD (and to some extent GSD and FGSD) find the best primal solutions, MS and MSS are generally off by 1% to 5%. As for each graph size 15 instances were considered and the heuristics were called (multiple) hundred times on I and II, we draw the conclusion that using MS(S) it is at least unlikely to come close to the *optimal* solution. Similarly, we can state that it is unlikely for FDR to generate solutions within less than 20% to optimality.

Lastly, Figure 12.7 depicts the improvement achieved in the objective by pruning Steiner nodes using Algorithm PruneSteinerNodes *for each found solution*. While the median improvement of the greedy diving heuristics is zero, the improvement exihibited especially by MS lie substantially higher. Interestingy, since MS and MSS achieve a very similar performance overall (cf. Figure 12.4), the leveling of their performance can be mainly explained by Algorithm's PruneSteinerNodes ability to reduce the objective value to *a greater extent* on solutions generated by MS than by MSS. This confirms the intuitive idea behind the MSS variant, since the selection of Steiner sites to activate is more stringent. Based on this observation, especially the runtime trade-off between using MS and pruning Steiner nodes and using MSS without pruning will be of interest.

12.3.3. Runtime

Figures 12.8 to 12.9 present the runtime of the heuristics excluding and including the runtime of Algorithm PruneSteinerNodes and Figure 12.10 depicts the runtime of Algorithm PruneSteinerNodes.

Considering the runtime excluding pruning Steiner nodes (see Figure 12.8), it can be observed that the runtime generally decreases according to the same order in which the quality of solution decreases. The one exception to this observation is the poerformance of FGSD on IGen instances having a substantially lower runtime than MSS. We nevertheless state the following observation.

Observation 12.3: The runtime of the LP-based heuristics generally decreases according to the same order in which the quality decreases, namely: GD, GSD, FGSD, MSS, MS, FDR.

Considering the runtime of the heuristics including pruning Steiner nodes (see Figure 12.9), the same observation holds as the runtime of Algorithm PruneSteinerNodes is nearly the same for all heuristics with the exception of FDR (see Figure 12.10). Importantly, the runtime needed to prune Steiner nodes is non-negligible as it takes e.g. on the largest IGen instances more than 40 seconds.

Considering the absolute runtimes without pruning Steiner nodes, we note that the runtimes seemingly grow exponentially with respect to the graph sizes (cf. Figure 12.8). While we do not extrapolate the corresponding factors, we note that this is to be expected at least on the 3D torus and fat tree topologies, as for the graph sizes I to IV the number of nodes is scaled by (at least) a factor of 2 (see Sections 10.3 and 10.4). Considering the IGen instances which grow linearily in the graph size, the runtime growth is considerably less. This allows for stating the following observation.

Observation 12.4: The experimentally measured runtimes of the LP-based heuristics support the theoretically obtained polynomial runtime bounds of Section 8.7.

Lastly, we notice that for some calls the runtime of GD, GSD and FGSD reaches the time limit of 1000 seconds on 3D torus instances of graph size IV and V. While not separately considered, this suggest that the respective heuristics' executions were aborted. Therefore, at least a fraction of the unsec:eval-heurisitcs-runtimesuccessful calls (see Figure 12.1) can be assumed to be due to exceeding the time limit and not due to the heuristic's inability to generate a solution.

12.4. Detailed Topology-Dependent Analysis

While in the above section the general performance and runtime trade-offs have been assessed, the heuristics' performance is now evaluated on a per topology basis. This will yield a (topology-dependent) selection of heuristics to include in the final VirtuCast solver.

12.4.1. Fat Tree

As established in Section 12.3.1 with the exception of FDR the heuristics always find solutions. Thus, by considering the minimal primal distance depicted in Figure 12.6, we can observe that GD finds high quality solutions *for all fat tree instances*. As the runtime of GSD and FGSD are not substantially lower (cf. Figure 12.8), the GD heuristic will be isec:eval-heurisitcs-runtimencluded in our final solver on fat tree instances. As the runtime of the MS(S) heuristic lies significantly below the runtime of the GD heuristic (cf. Figure 12.8), while still providing solutions off by 5% of the best primal solution (cf. Figure 12.4), these will also be included. Thus, in case GD fails to produce a solution, a good solution might be found using multiple executions of the MS(S) heuristics. Since the runtime of MS lies significantly below the runtime of MSS for graph sizes IV and V, the MS heuristic will be applied on these instances.

12.4.2. IGen

Considering the heuristics' performance on IGen topologies, we first note that both the quality of found solutions as well as the runtimes are equally distributed across all graph sizes (cf. Figures 12.4 and 12.8). As again GD provides the highest quality solutions (cf. Figure 12.4

while the (F)GSD and MS(S) heuristics exhibit a similar runtime (cf. Figure 12.8) and the runtime on even the largest instances generally lies below 60 seconds, the GD heuristic will be employed on IGen instances.

As the runtime of the other heuristics when pruning is applied does not lie substantially lower (cf. Figure 12.9), the MS heuristic will be used on IGen instances without pruning Steiner nodes. Using the MS instance in such a way is based on the distinct runtime advantage obtained without pruning (cf. Figure 12.8.

By a set of similar observations, the MS heuristic is selected to be included, but without pruning Steiner nodes via Algorithm PruneSteinerNodes. This decision is justified by the observation that MS still comes reasonably close to the best primal solution even without pruning (cf. Figure 12.5).

12.4.3. 3D Torus

The performance of heuristics on 3D torus topologies the most complex to assess, as the runtime of the greedy diving heuristics exceeds or comes close to the time limit of 1000 seconds (see Section 12.3.3). As in our final experiments a time limit of one hour is enforced, employing the greedy diving heuristics GD and GSD on graph sizes IV and V may therefore severely limit both the progress of the dual bound and the time left for finding solutions if these heuristics fail to produce one.

Considering graph sizes I, II and III, we note again that the GD heuristic comes the closest to the best primal solution found (cf. Figures 12.5 and 12.6), while runtimes are very similarly distributed (cf. Figure 12.8). Furthermore note that the median performance achieved by the MSS heuristic without pruning comes close to the performance of GD. On these instances therefore both GD and MSS will be employed.

For graph size IV of 3D torus instances, we note that for the first time FGSD provides a distinct runtime improvement over GD and GSD (cf. Figure 12.8 while the quality of the solutions generated is only off by at most 0.3% with respect to the objective of the best solution found without pruning Steiner nodes (cf. Figure 12.5). Therefore FGSD will be employed. As MSS still provides a lower runtime if pruning is enabled compared to the runtime of FGSD when puning is disabled (cf. Figures 12.8 and 12.9) and as MSS yields solutions only 2% off the best primal solution found (cf. Figure 12.6), MSS is included on graph size IV.

Considering the largest graph size, FGSD is included by the same argument used for including it on graph size IV. However, as the runtime of MSS surpasses the runtime of FGSD on this graph size (cf. Figure 12.8), instead of MSS the MS heuristic is employed. Since the improvement of the quality of the solutions found by MS lies around 1% (cf. Figure 12.7), Algorithm PruneSteinerNodes will be utilized to try to improve the solutions' quality.

12.4.4. Chosen Parameter Settings

In the above sections the performance of the LP-based heuristics have been discussed for each topology and a selection of heuristics were chosen to be used for the different graph sizes. This selection of heuristics is summarized in Table 12.2 and enriched with frequency and offset values which togehter determine the depth levels at which the heuristics are called.

Given an offset O and a frequency F, heuristics are called at each node of the following depth levels $\{O, O+F, O+2F, O+3F, \dots\}$, where depth level 0 denotes the root level of the branch-and-bound tree.

Even though we will not discuss the offset and frequency settings in detail, we note that by utilizing an offset of greater 0 increases the chances that the heuristic will be called more than once initially. Furthermore, by setting the frequency to 0, the heuristic will only be called at the depth level defined by the offset.

		GD			**GSD**	**FGSD**			**MSS**			**MS**			**FDR**		
		O	F	P	-	O	F	P	O	F	P	O	F	P	O	F	P
FAT TREE	I-III	0	20	×	-		-		2	10	✓		-			-	
	IV-V	0	0	×	-		-			-		5	10	✓	0	0	✓
IGEN	I-V	0	10	×	-		-			-		2	20	×		-	
3D TORUS	I-III	2	10	×	-		-		5	15	×		-			-	
	IV		-		-	2	10	×	5	15	✓		-			-	
	V		-		-	2	10	×		-		5	15	✓		-	

Table 12.2.: Selection of heuristics for final VirtuCast solver. The following abbreviations are used. O: offset, F: frequency, P: apply PruneSteinerNodes

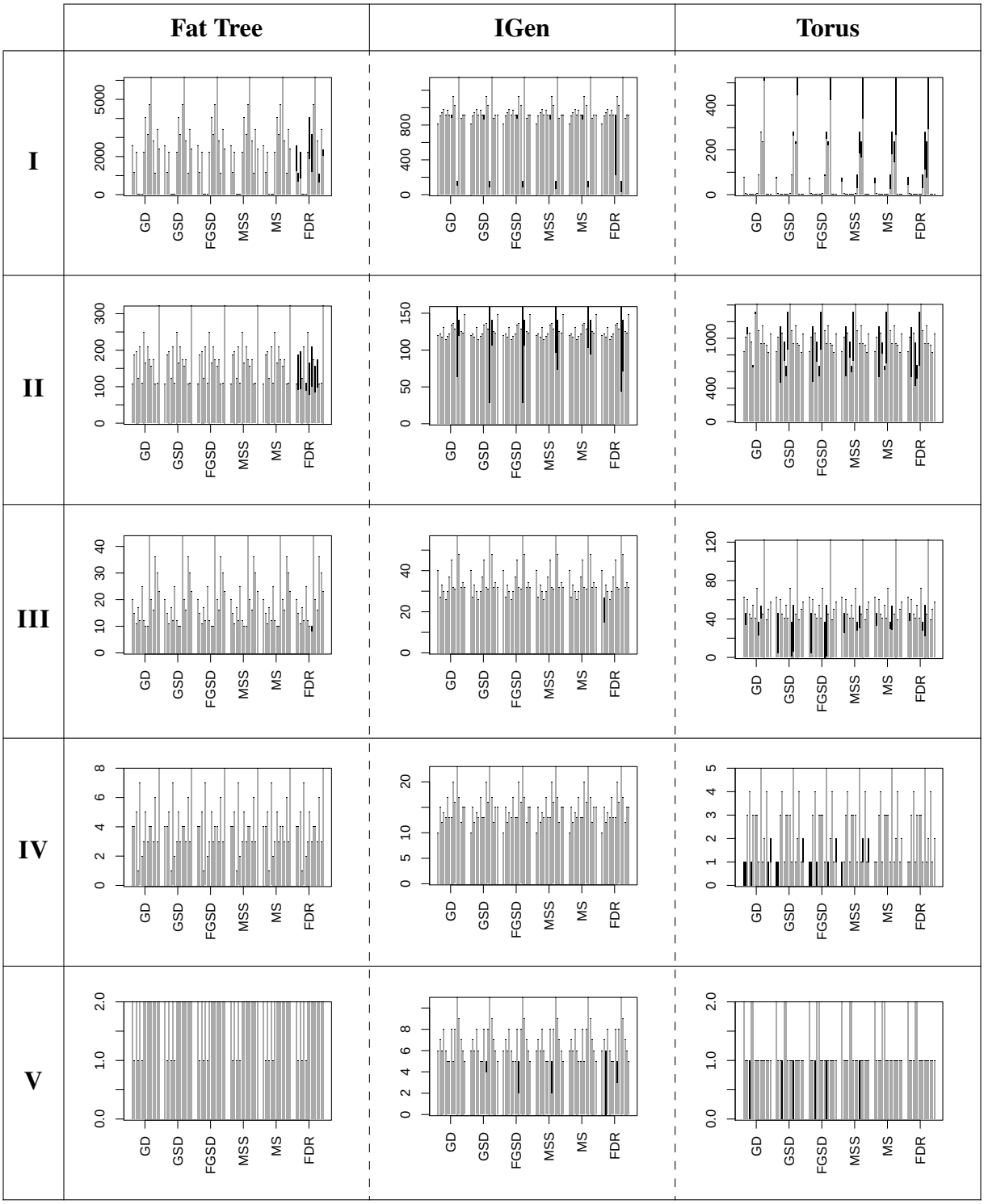


Figure 12.1.: Number of the (un)successful calls to the heuristic *for each instance*. The grey bar indicated the number of successful calls that yielded a feasible solution. The black bar indicates the number of calls in which no solution could be obtained.

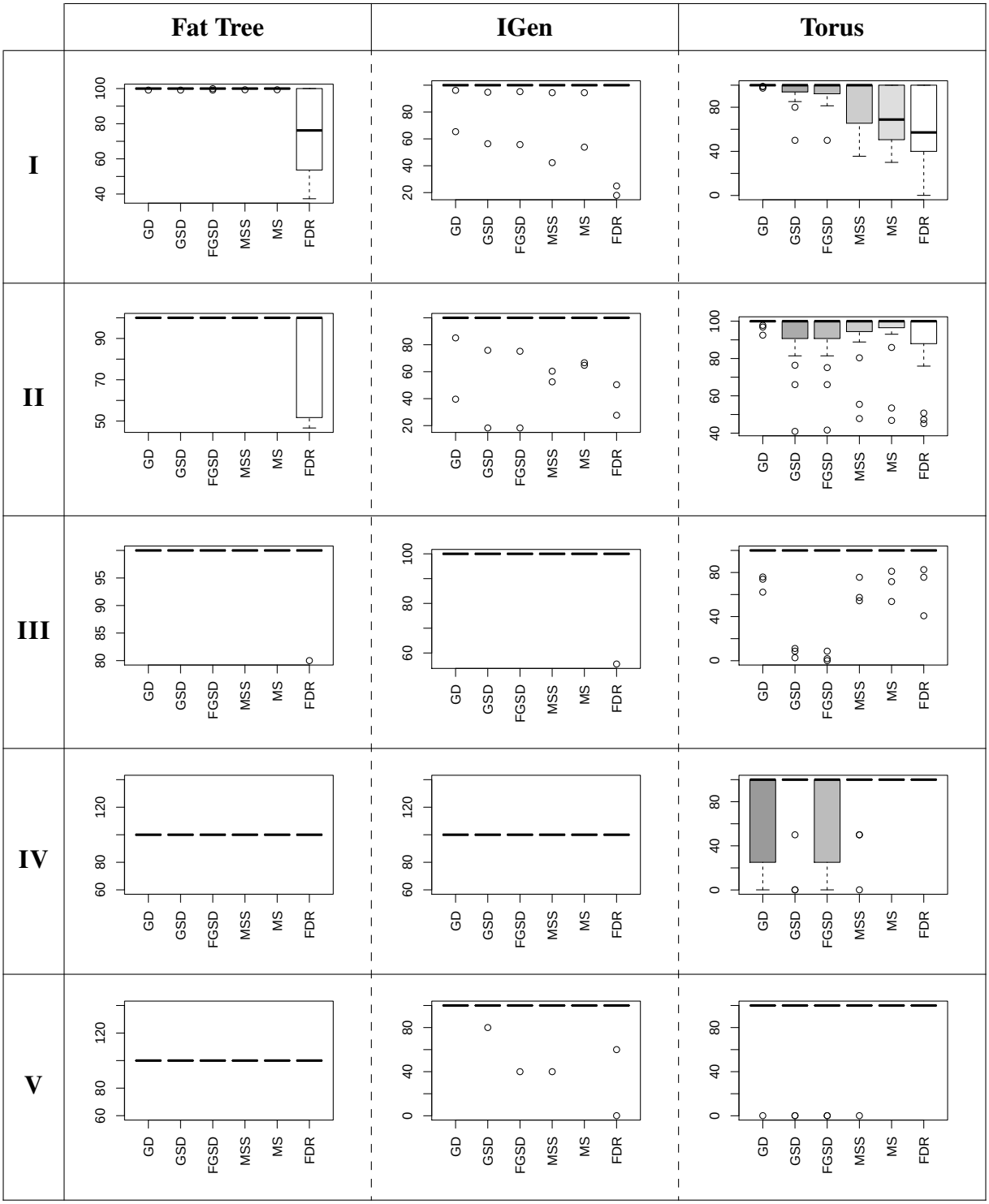


Figure 12.2.: Frequency (in percent) with which the LP-based heuristics have found solutions.

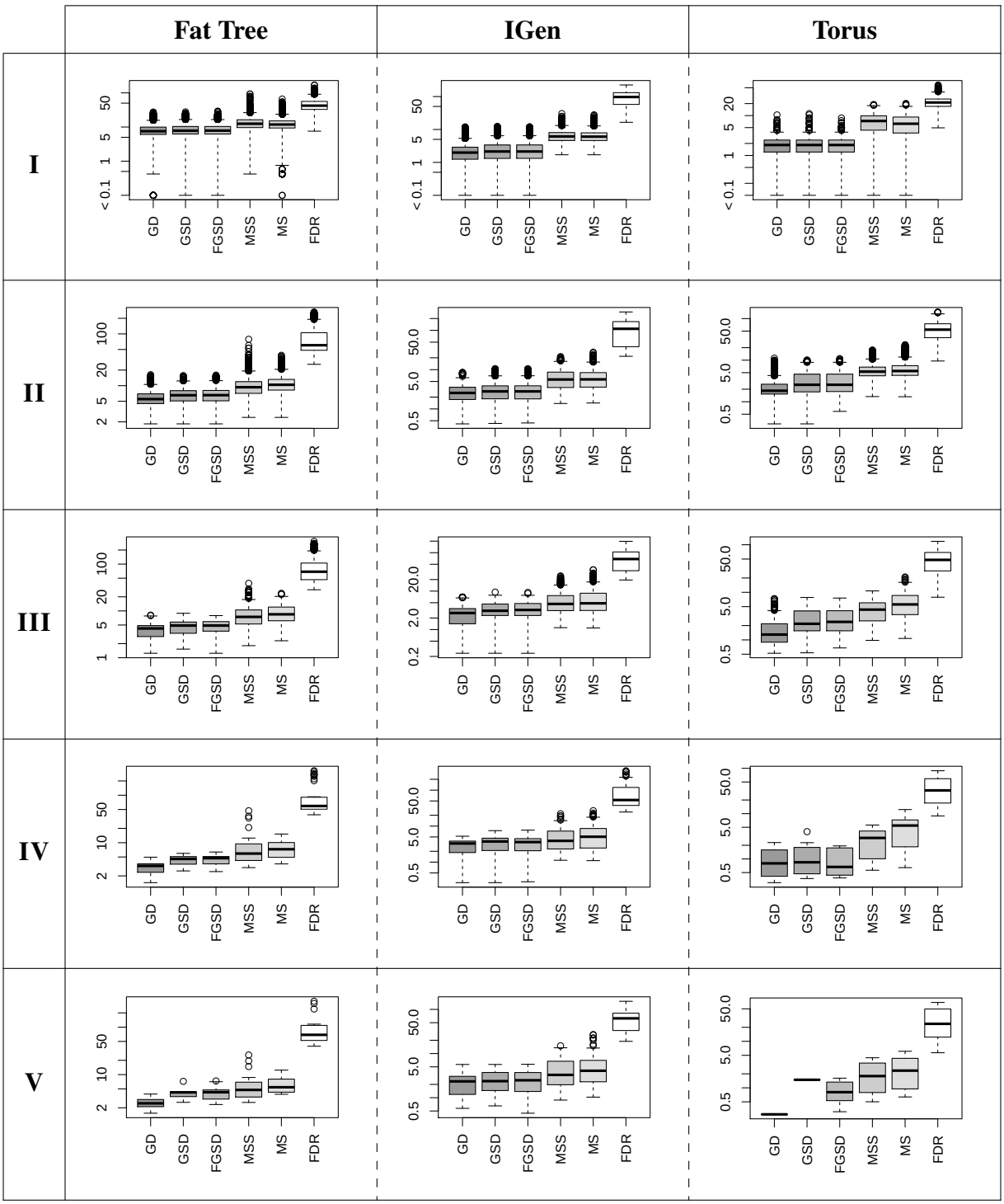


Figure 12.3.: Objective gap (in percent) with respect to the final dual bound after Algorithm PruneSteinerNodes was called *per call, if a solution was found*.

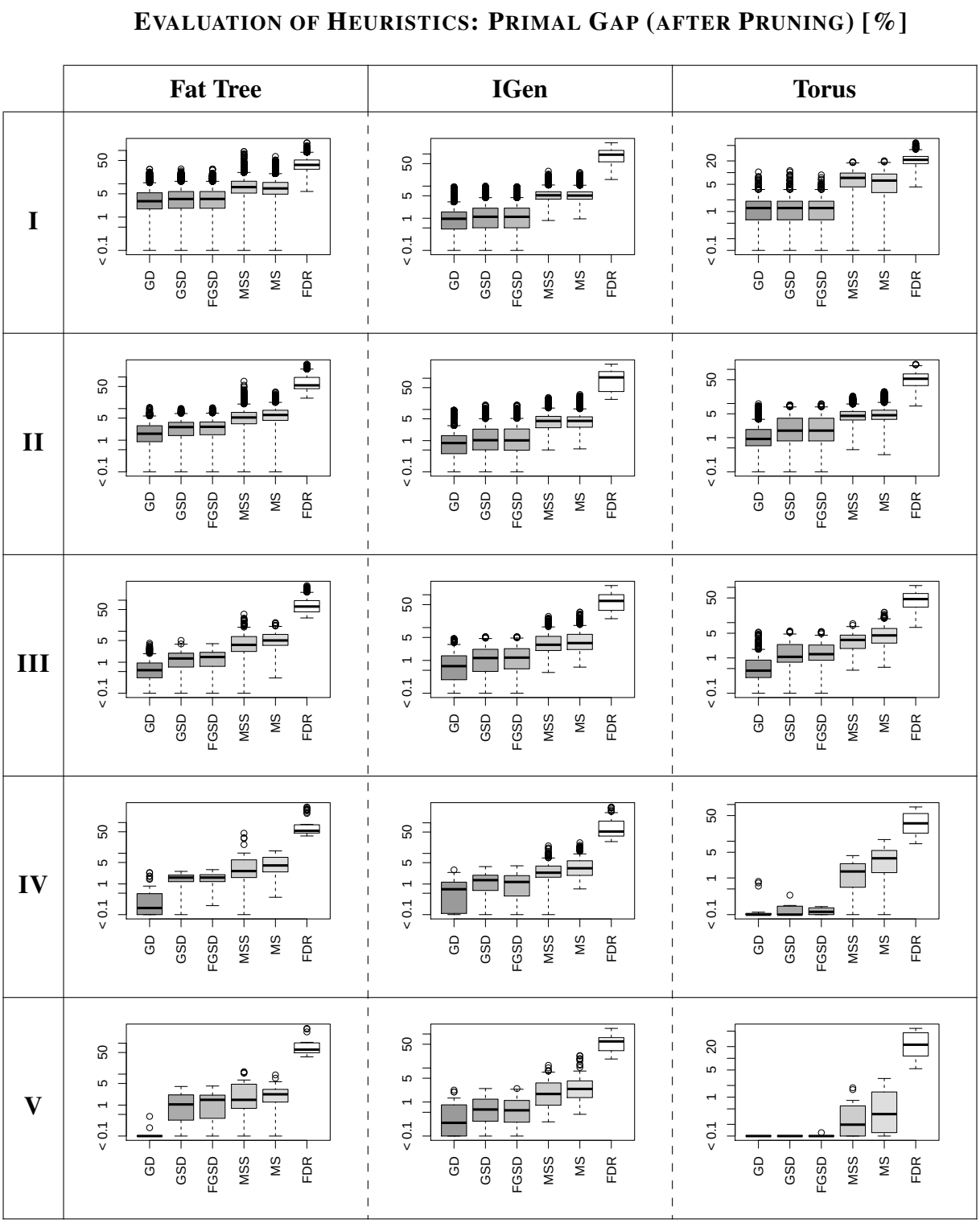


Figure 12.4.: Objective gap (in percent) after Algorithm PruneSteinerNodes was called with respect to the best heuristical found primal solution.

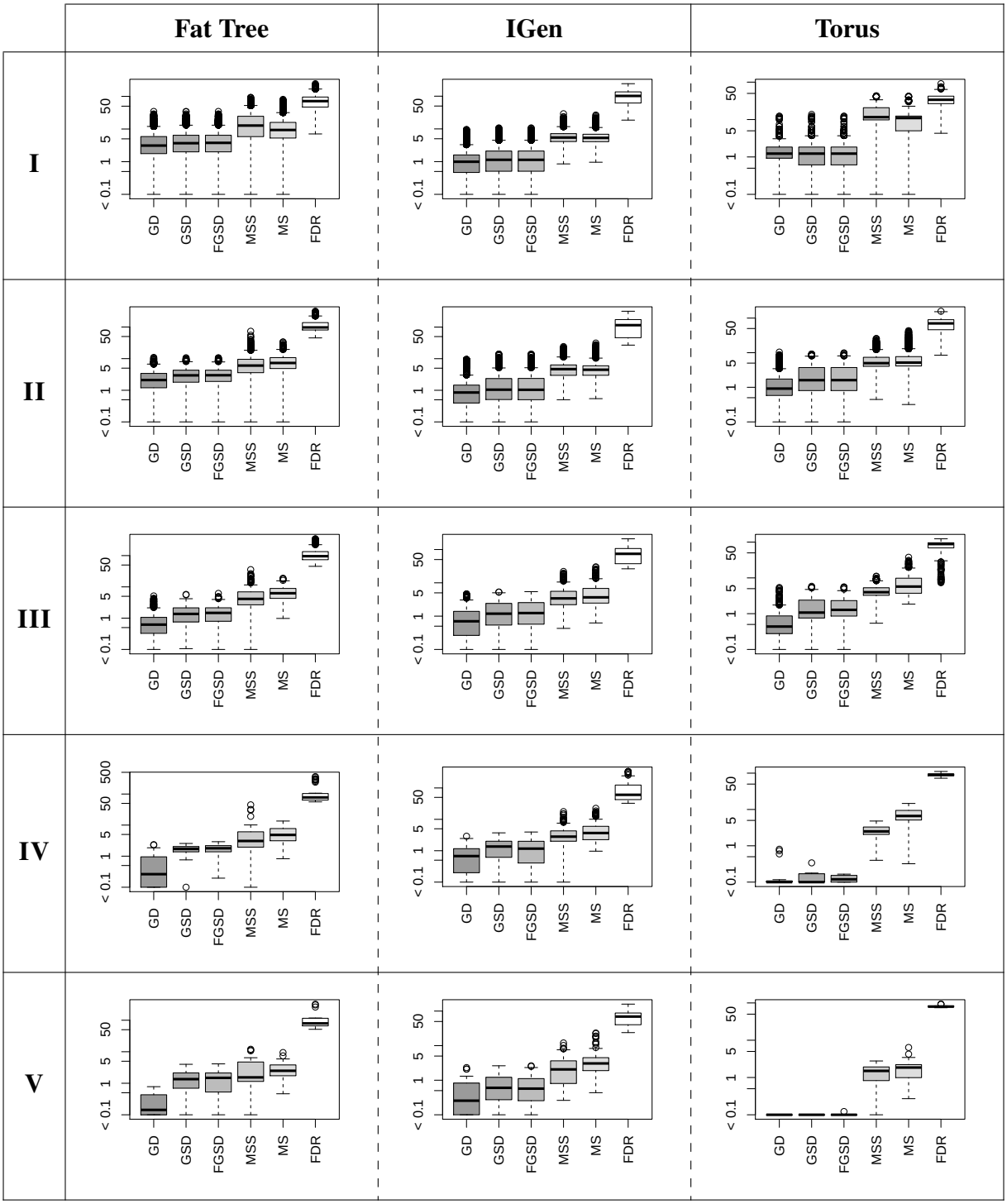


Figure 12.5.: Objective gap (in percent) before Algorithm PruneSteinerNodes was called with respect to the best heuristical found primal solution.

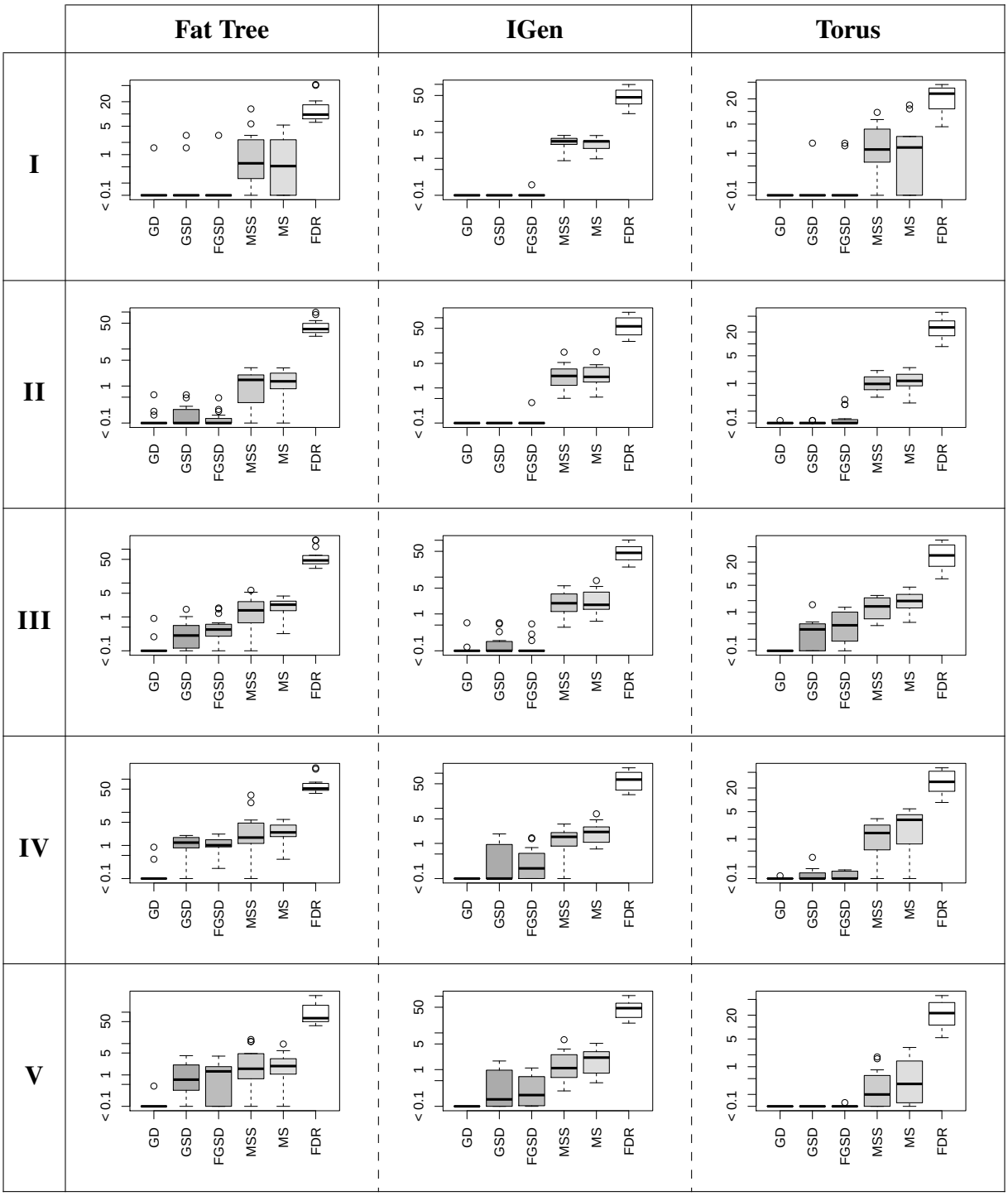


Figure 12.6.: Minimal objective gap (in percent) after Algorithm PruneSteinerNodes was called with respect to the best heuristical found primal solution that was obtained by the heuristics *per instance*.

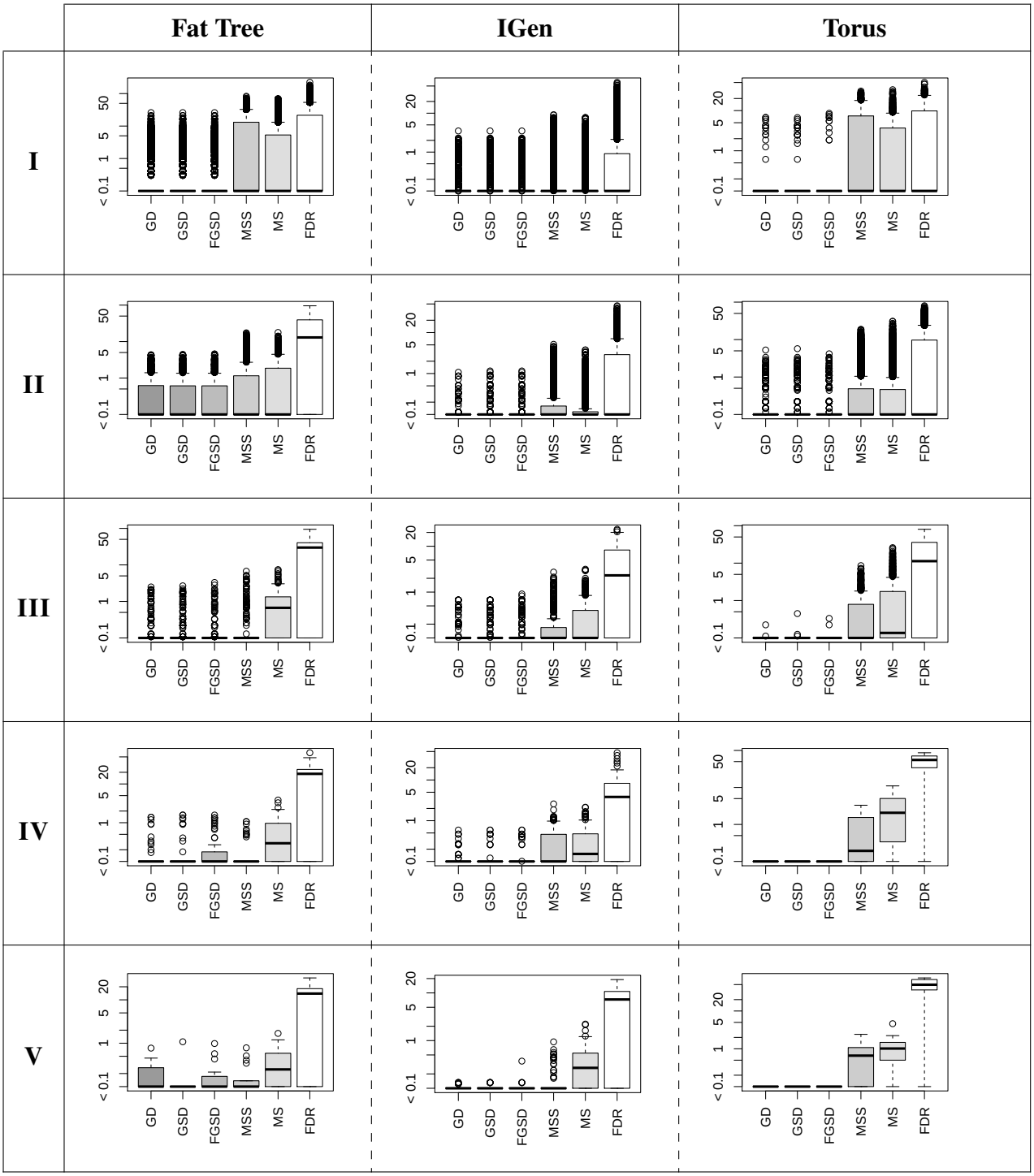


Figure 12.7.: Improvement of the objective gap (in percent) with respect to the final dual bound that is achieved by calling PruneSteinerNodes *per call, if a solution was found.*

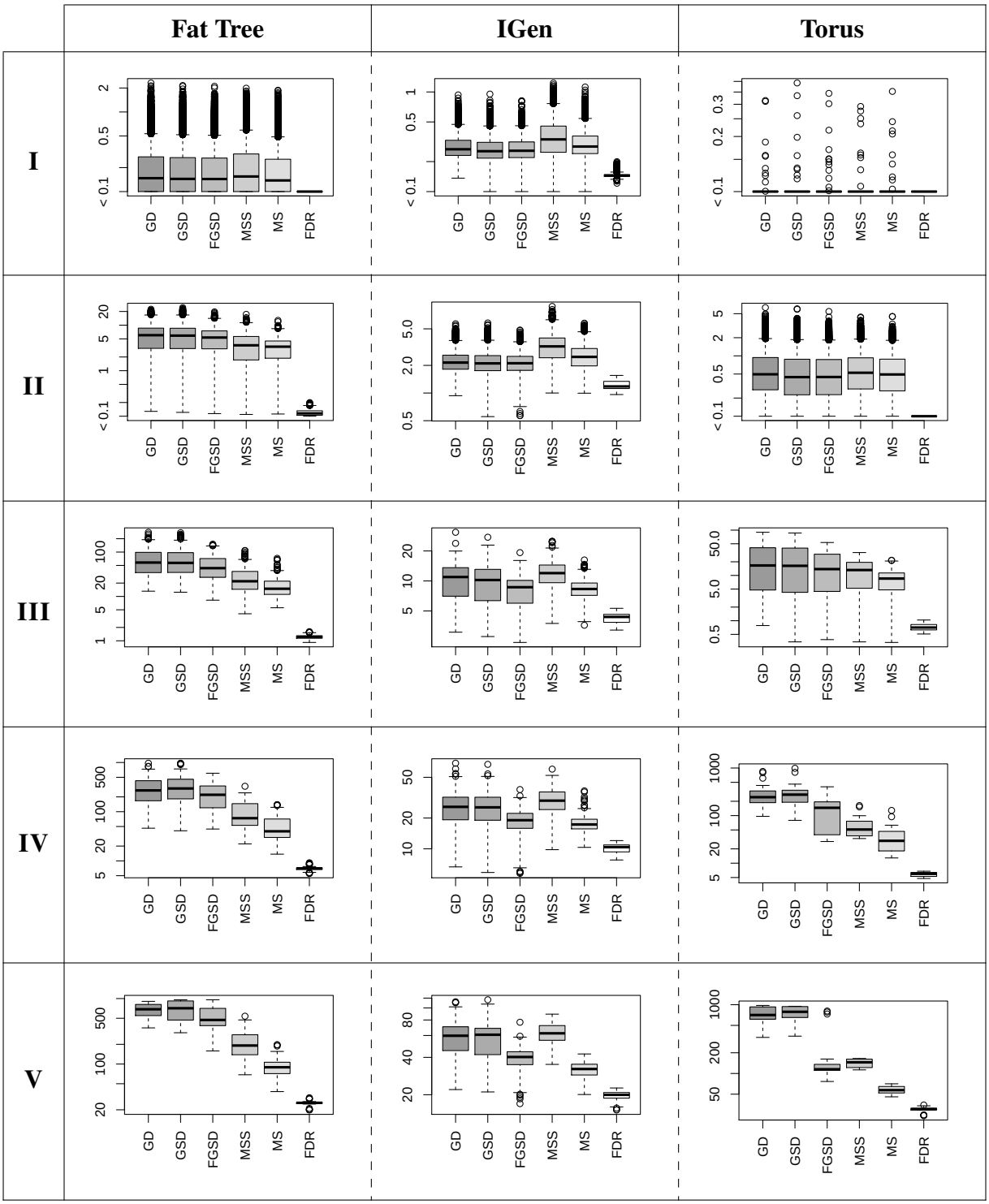


Figure 12.8.: Runtime in seconds of the heuristics without the time spent in PruneSteinerNodes *per call, even if no solution was found.*

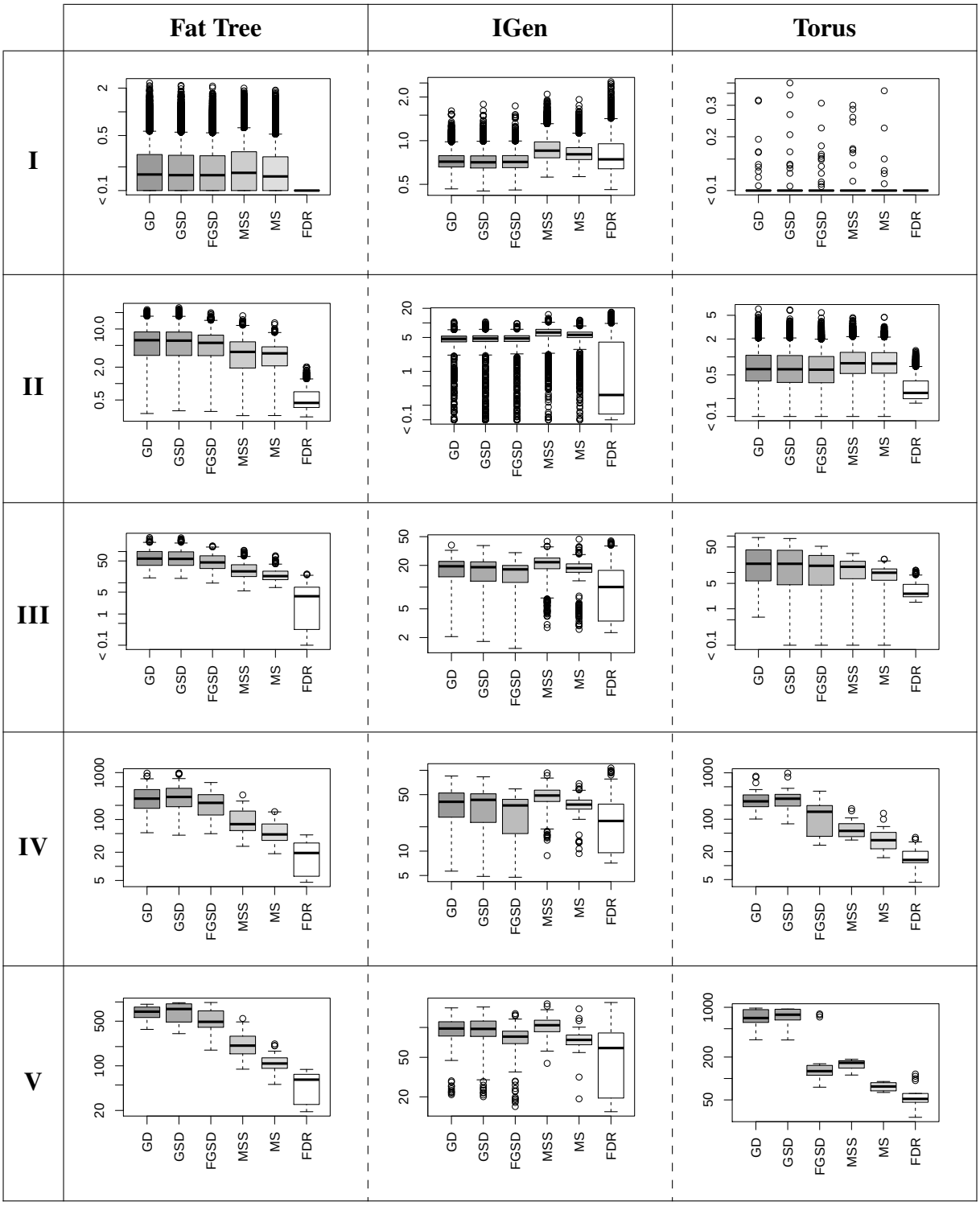


Figure 12.9.: Runtime in seconds of the heuristics including the time spent in PruneSteinerNodes.

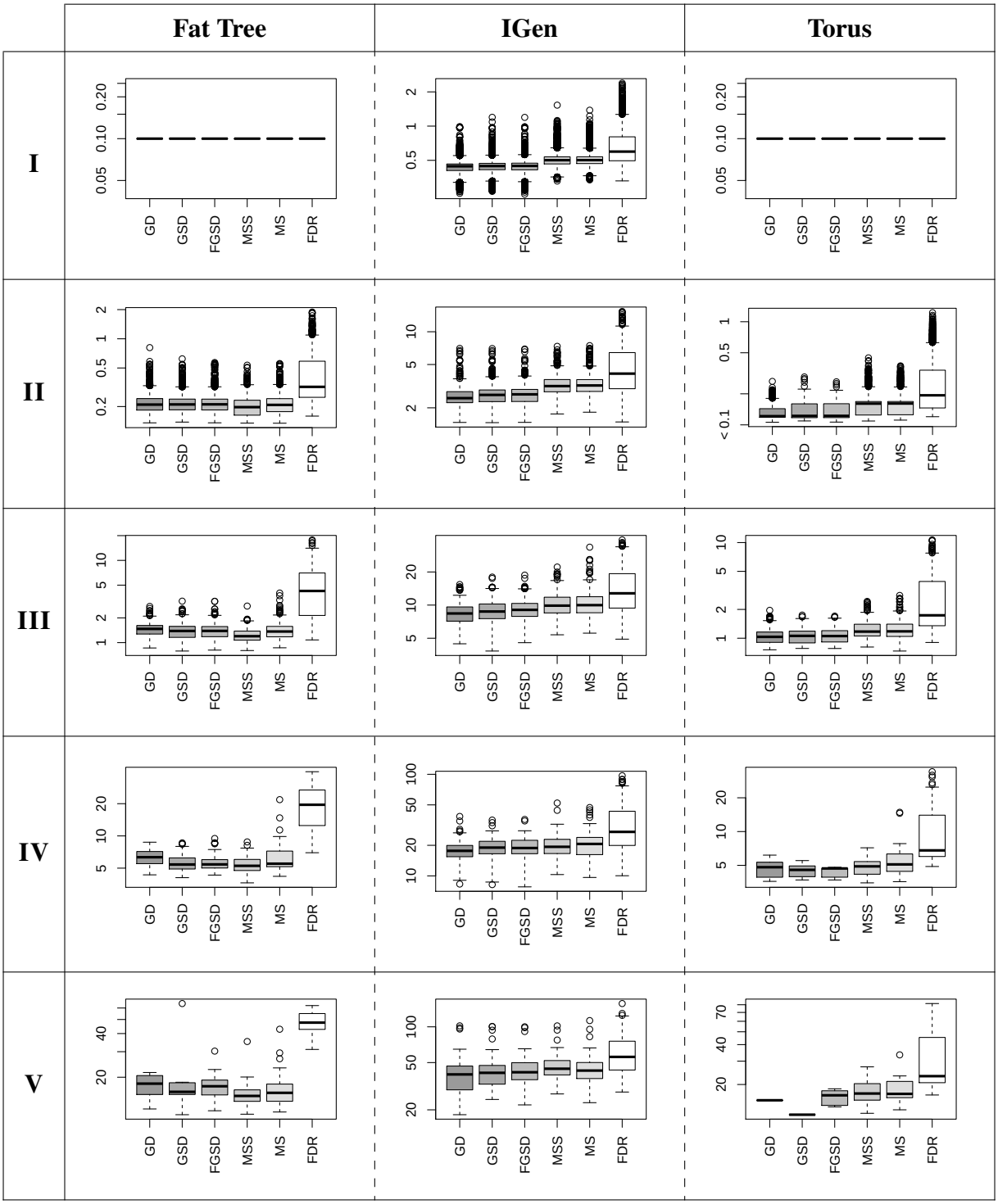


Figure 12.10.: Runtime in seconds of Algorithm PruneSteinerNodes *per call, if a solution was found.*

13. Performance of the Final VirtuCast Solver

In this section the performance of the final VirtuCast solver is presented. For this set of experiments the separation and branching parameters that were chosen in Section 11.5 are used. Furthermore, depending on the topology and the graph size, LP-based heuristics are included as presented in Section 12.4. Importantly, all other SCIP heuristics are disabled such that solutions obtained have been generated by the LP-based heuristics or are obtained via integral LP relaxations. Experiments are conducted on all the 225 instances as discussed in Section 10.2. However, the instances used for this evaluation are *generated independently* from the instances used in the previous, preliminary evaluations of Sections 11 and 12. Hence, obtaining high quality solutions in this set of experiments *validates* the choice of heuristics and cannot be attributed to 'learning' the right selection of heuristics beforehand.

Figure 13.1 presents the main performance characteristics of the final VirtuCast solver. For the smallest graph sizes more than half of the instances can be solved to optimality. Considering the other graph sizes, the median objective gap on IGen and 3D torus instances is less than 1.5%, with the maximal gap being less than 5%. In contrast, on fat tree instances the median objective gap lies within 3% to 5%, with the maximal objective gap of slighlty less than 6% being observed on an instance of graph size II.

With respect to the improvement of the dual bound, we note that on graph sizes III to V only a negligible improvement of less than 0.2% can be observed. For the smallest graph size, the median improvement on fat tree and 3D torus instances lies at 3% and 2.5%. Given these relatively small improvements in the dual bound, and by adding the dual bound improvement to the achieved objective gap per graph size, the following observation can be made.

Observation 13.1: The objective value of the optimal solution *always* lies within at most 6% of the root's dual bound.

In Figure 13.1 also the overall runtime of the heuristics is presented. Note that the heuristics, consume less than half an hour of runtime. Thus, the selection of frequencies (and offsets) still allows for progress in the branch-and-bound tree while achieving near optimal solutions.

Lastly, we note that it may be possible to further improve the performance of the VirtuCast solver by using more advanced validation tools as cross-validation for finding the best combination of separation, branching and heuristic settings. Furthermore, as all of SCIP's heuristics were disabled, certain (IP-based) local search heuristics (e.g. crossover, mutation and oneopt) to improve the objective value of solutions' found are disabled as well. Enabling these might allow to further reduce the objective gap.

PERFORMANCE OVERVIEW OF FINAL VIRTUCAST SOLVER

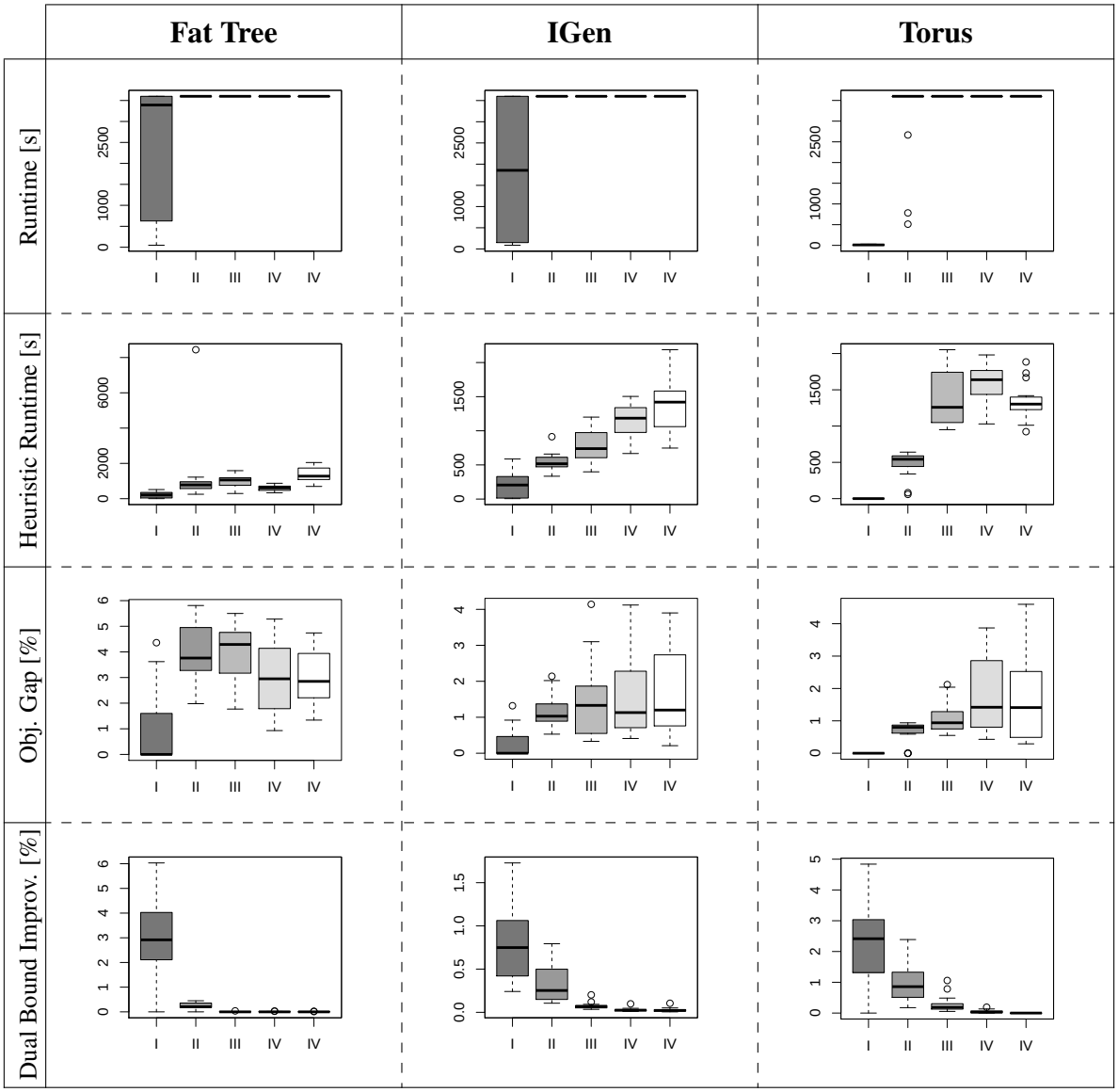


Figure 13.1.: Overview of the performance of the final VirtuCast solver on the different topologies.

14. Evaluation of alternative Algorithms

Having discussed the results of the final VirtuCast solver in Section 13, the performance of alternative solution approaches is discussed in this section. The alternative solution approaches considered are utilizing the GreedySelect heuristic (see Section 14.1), solving IP formulation A-CVSAP-MCF (see Section 14.2) and using the VirtuCast solver with SCIP's heuristics only (see Section 14.3).

14.1. GreedySelect

We begin by considering the GreedySelect heuristic, which was implemented in C++ (as part of the VirtuCast solver). As the heuristic is of a combinatorial nature and does not use randomization, the heuristic is called exactly once for all the instances considered in Section 13. The heuristic's performance across the different topologies is depicted in Figure 14.1.

As for each graph size fifteen instances are considered, we first note that the heuristic finds a feasible solution on all fat tree instances, on nearly all IGen instances and fails to produce more than one solution of 3D torus instances. As on 3D torus instances at most a single solution is found, the singular results on this topology will not be discussed.

Figure 14.1 furthermore depicts the objective gap with respect to the best dual bound established in the experiments of Section 13. While on fat tree instances the median objective gap after pruning lies around 40%, the median objective gap (for the instances on which solutions were found) on IGen instances drops from around 17% to below 10% on larger graph sizes. Importantly, to obtain solutions with an objective gap below 100% on fat tree instances, the usage of Algorithm PruneSteinerNodes is necessary. In contrast, on IGen instances an objective gap below 30% can be observed, when no Steiner nodes are pruned.

The runtime of the GreedySelect heurisitc is subdivided into the runtime of the heuristic itself and the runtime consumed by Algorithm PruneSteinerNodes. Similar to the exponential growth of runtimes of the LP-based heuristics (cf. Figure 12.8), the runtime of Algorithm GreedySelect grows nearly exponentially in the graph size. While this does not contradict the polynomial runtime (see discussion in Section 12.3.3 and Observation 12.4) no distinct runtime advantage can be observed, as for the LP-based heuristics generally a very similar runtime can be observed on both fat tree and IGen instances (cf. Figure 12.8).

While the above presented results must not be seen as a general indicator for the inherent weakness of combinatorial heuristics, they show that the runtime performance of the LP-based heuristics is actually competetive. Even though the runtime of GreedySelect could be substantially reduced by only considering a fraction of inactive Steiner nodes to open,

combinatorial heuristics may still fail to reliably generate feasible solutions without allowing for path reconfigurations.

14.2. Performance of Formulation A-CVSAP-MCF

The main results of the evaluation of the IP formulation A-CVSAP-MCF, which was solved using CPLEX, are presented in Figure 14.2. Note that no results for graph size V are presented. This is due to the fact that GLPK (see Section 4.3 for the 'implementation') fails to generate executable .lp files for graph sizes of V as these require multiple gigabyte of storage. Nevertheless, the results obtained on graph sizes I to IV allow to draw qualitative conclusions regarding the applicability of formulation A-CVSAP-MCF. CPLEX was run using default parameters, making all of the 16 GB of physical RAM available to it. As for the VirtuCast experiments of Section 13, we limit the runtime to one hour.

In the top two rows of Figure 14.2 the number of instances for which a solution was found and the number of instances for which the root relaxation could be computed are depicted. Importantly, due to the number of variables and constraints required in the multi-commodity flow formulation, CPLEX fails to solve the root relaxation on multiple fat tree instances of graph sizes III and IV and on some 3D torus instances of graph size IV. While this does not suffice to computationally substantiate the weakness of the formulation, it shows the inapplicability to *solve* instances of these topologies and graph sizes *within reasonable time*.

To establish the computational weakness of formulation A-CVSAP-MCF and vice versa argue for the computational strength of formulation IP-A-CVSAP, the dual bound gap is also depicted in Figure 14.2. In this case, the dual bound gap is computed by $(D_{VC} - D_{MCF})/D_{VC}$, where D_{VC} denotes the best dual bound obtained by the VirtuCast solver and D_{MCF} denotes the best dual bound obtained by solving formulation A-CVSAP-MCF. Thus, the dual bound gap measures the relative distance of the A-CVSAP-MCF's dual bound to the dual bound established by IP-A-CVSAP. While the dual bound gap on 3D torus instances is relatively small, its median lies around 5% on IGen and between 9% and 20% on fat tree instances (for the instances where a dual bound could be established by CPLEX). This is in stark contrast to Observation 13.1, namely that the relative distance of the optimal solution's objective to the root relaxation lies below 6% when using formulation IP-A-CVSAP. We therefore conclude, that formulation IP-A-CVSAP is indeed (computationally) stronger than formulation A-CVSAP-MCF.

14.3. VirtuCast with SCIP's Heuristics

Lastly, to motivate the development of the LP-based heuristics defined in Section 8, experiments have been conducted using the VirtuCast solver, such that all of SCIP's heuristics are enabled, while none of the LP-based heuristics of Section 8 are included. For ensuring comparability, experiments were run on the same set of instances as used in Section 13 and using the same separation and branching parameters. Additional to including all of SCIP's heuristics, the heuristic setting aggressive is used, such that the frequency of heuristics is halved and

heuristics are called twice as much. Again, the runtime limit is set to one hour.

Figure 14.3 presents the results obtained by using the VirtuCast solver in combination with SCIP's heuristics. Considering the runtime consumed by SCIP's heuristics, we observe a similar runtime allocation as during the final VirtuCast solver on fat tree instances, while on IGen instances the runtime spent in heuristics is lower and on 3D torus instances exceeds the runtime using the final VirtuCast solver (cf. Figure 13.1.

Considering the objective gap that is achieved, it can be observed that for graph sizes I and II a similar performance is obtained (cf. Figure 13.1), while beginning with graph size III the objective gap considerably worsens. For instances of graph size IV and V, for most instances no solution can be obtained anymore.

With respect to the improvement of the dual bound (see Figure 14.3), we observe a slight, but general, performance degradation across all topologies and instances (cf. Figure 13.1).

Without performing an in-depth analysis of which heuristics were most effective during the experiments, we note that SCIP's default heuristics fail to reliably generate solutions across all topologies. Furthermore, if solutions are found for instances of graph size IV these solutions may exhibit objective gaps as large as 100%.

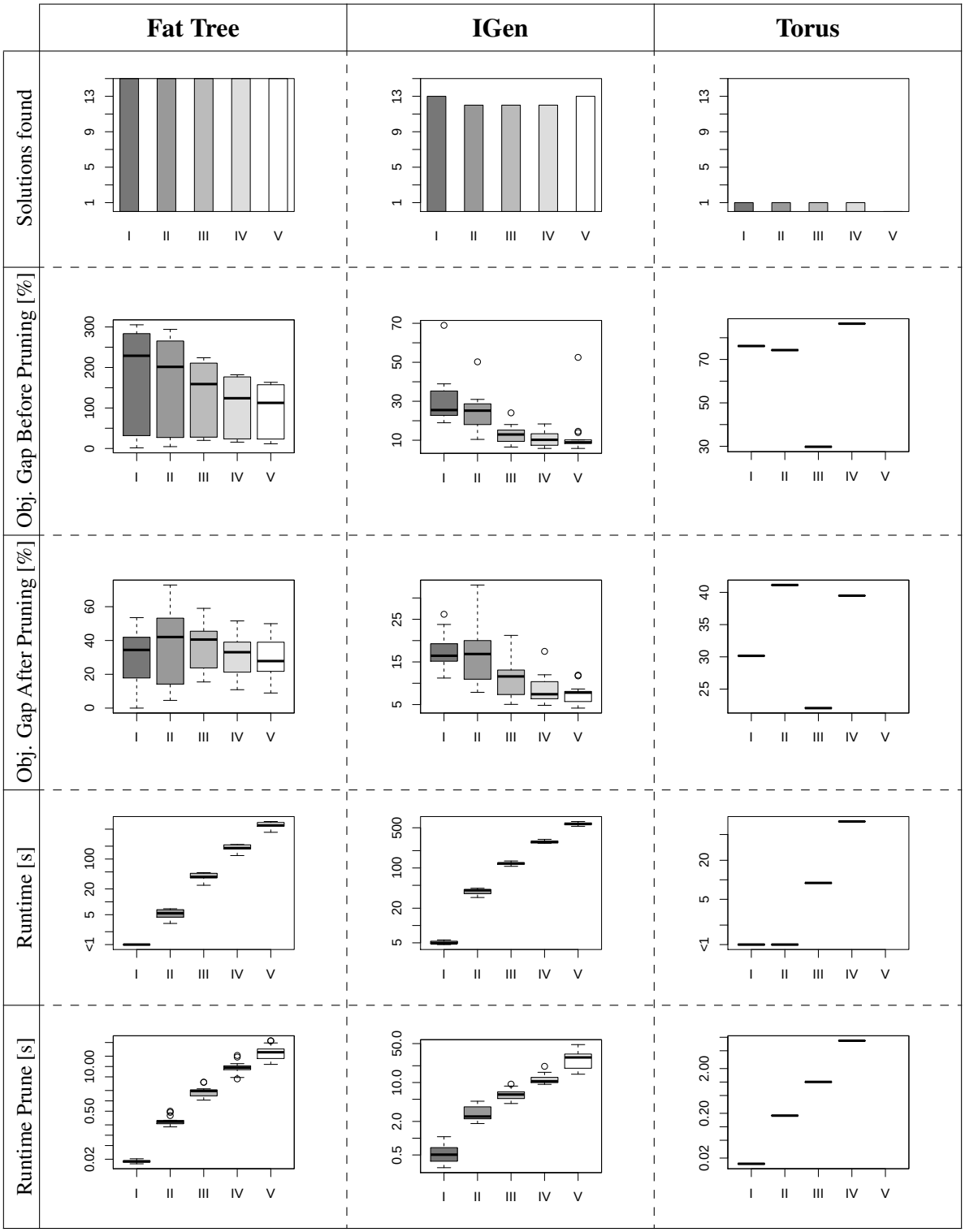


Figure 14.1.: Performance of the GreedySelect heuristic on all instances that were considered in Section 13.

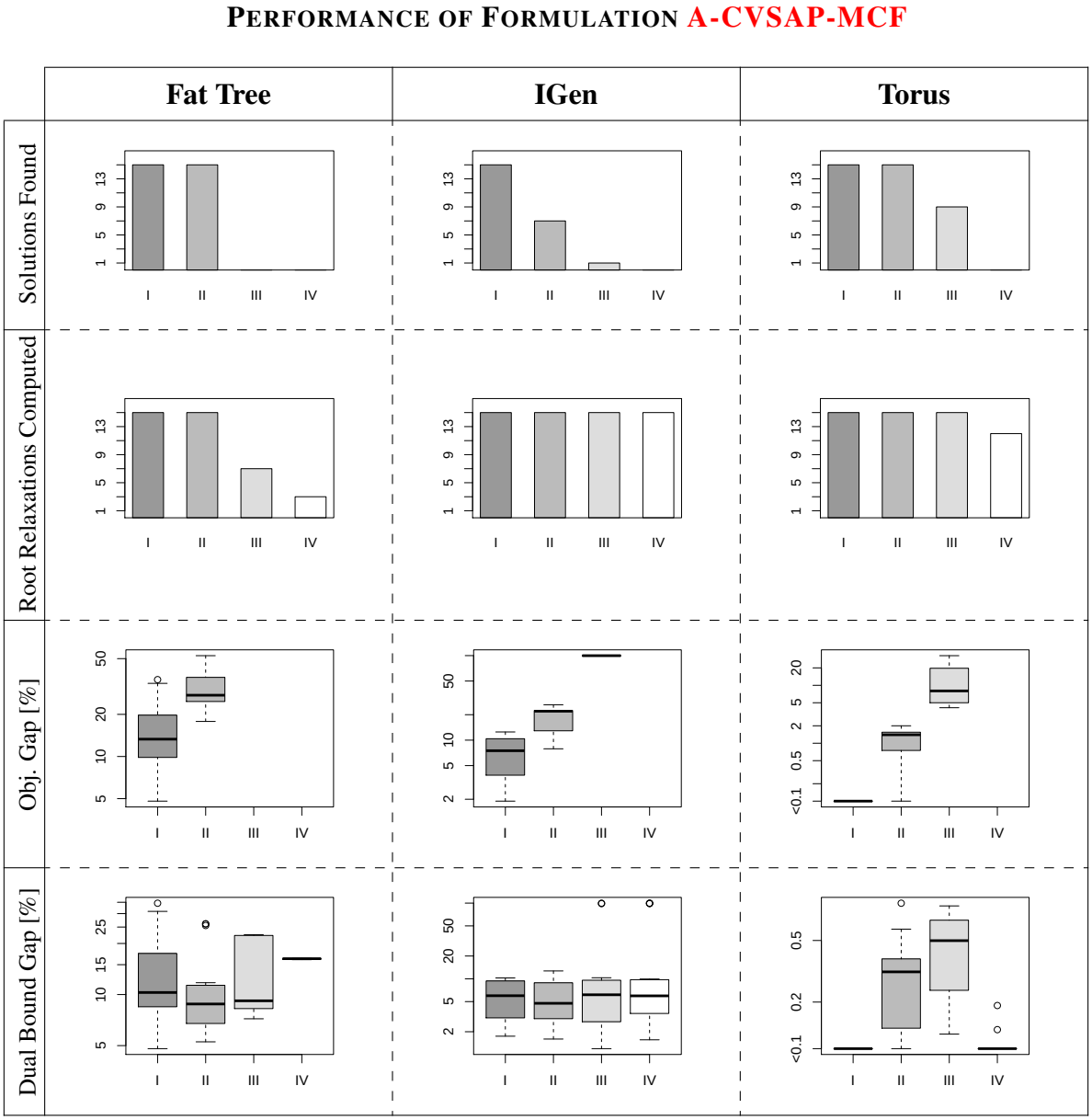


Figure 14.2.: Performance of formulation A-CVSAP-MCF, which is solved using CPLEX, on the instances of graph size I-IV that were considered in Section 13.

PERFORMANCE OVERVIEW OF VIRTUCAST WITH SCIP'S HEURISTICS

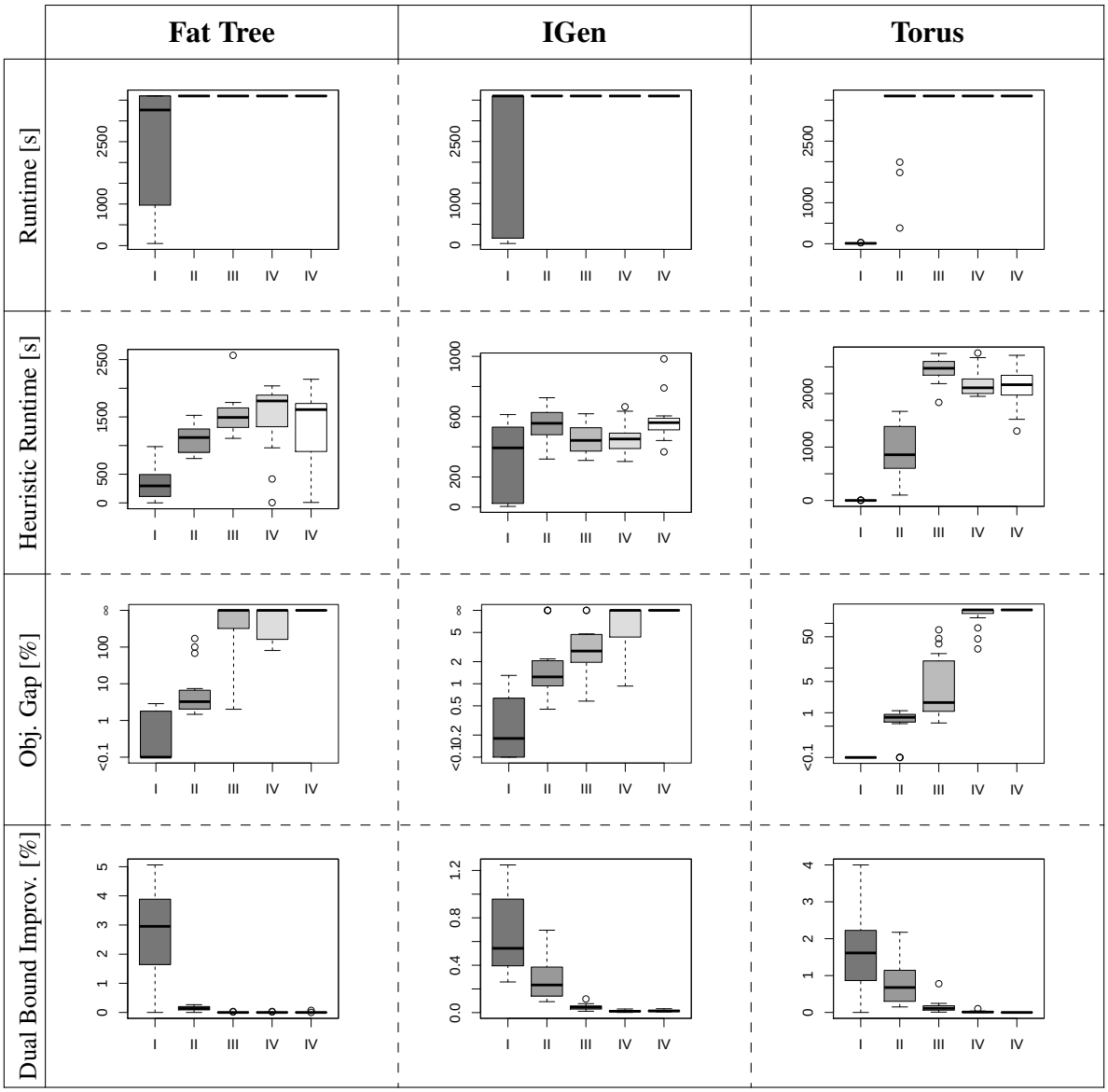


Figure 14.3.: Overview of the performance of the VirtuCast solver *without* the LP-based heuristics presented in Section 8, but *with* all of SCIP's heuristics activated, on the instances that were considered in Section 13.

Part V.

Conclusion

15. Related Work

In this section both theoretical and practical related work is shortly summarized. One very important theoretical result we want to highlight was shown by Molnar in [**molnar2011hierarchies**]. In his work, Molnar considers classes of spanning tree problems and their underlying solution structure. Without deriving algorithms for solving the considered problems, Molnar proves that (directed) acyclic graphs are not sufficient to solve these problems. This result also applies to CVSAP and underlines the necessity of using the novel Algorithm Decompose to obtain paths, as the result of any single-commodity flow formulations may not be acyclic.

In the following in Section 15.1 related work considering the applicability of CVSAP is summarized, that was not discusses before. In Section 15.2 further connections to other optimization problems are mentioned. We conclude in Section 15.3 by pointing towards literature related to Integer Programming formulations that were not covered before.

15.1. Applicability of CVSAP

The conceptually most related works are the ones of Banerjee et al. on efficient overlay multicast networks [Ban+03] and the Flow Streaming Cache Placement Problem of Oliviera and Pardalos (see discussion in Section 1.2.1). Banerjee et al. consider the distribution of multicast stations in an existing network to deploy overlay multicast functionality [Ban+03]. As they consider both flexible processing locations and degree-bounds their model is closely related to CVSAP. They propose an Integer Programming formulation in which however all nodes must be connected and therefore no flexibility is included.

In different contexts, the observation was made that multicasting and aggregation can be efficiently implemented by utilizing only a subset of nodes for in-network processing. For example Shi showed in [Shi01] the general applicability of selecting only a few processing nodes for multicasting. Similarly, [KVM02] considered overlay networks for multicasting and arrived at the conclusion that even by randomly placing processing functionality on only a few nodes the traffic is only increased by 50%. In the context of multicast routing in the optical domain [PLK10] considers sparse-splitting networks and shows that already few multicast-enabled optical routers suffice. With respect to the applicability of our model to all-optical multicast, we note that Rouskas specifically considers energy loss when wavelengths are distributed using passive splitters [Rou03]. Based on the limited number of outgoing connections we allow for in the multicast variant, the selection of splitting locations can be modeled using CVSAP.

15.2. Related Theoretical Problems

Besides the connections to Connected Facility Location, the Steiner Arborescence Problem and the Degree-Constrained Node Weighted Steiner Tree Problem that have been established in Section 3, there are several other related problems.

Segev first introduced the Node-Weighted Steiner Tree Problem [Seg87], which is closely related to Prize-Collecting Steiner Tree Problems (PCSTP). Segev formulated the problem in the following way: given a set of terminals that must be connected, negative costs for non-terminal nodes and positive edge costs, find a Steiner Tree of whose cost, i.e. the sum of node and edge costs which are included, is minimal.

Several different variants of the PCSTP are considered by Johnson et al. in [JMP00]. The original'Goemanns-Williamson' problem formulation asks for a minimal cost Steiner Tree where nodes not included increase the cost while in contrast the net worth maximization variant asks for finding a Steiner Tree such that for each included node a (beneficial) prize is paid, while using edges within the Steiner tree induces costs. Another interesting variant is the budget variant, in which a Steiner Tree of (edge) costs less than a certain budget shall be constructed, that maximizes the prizes collected for connected nodes. Note that the formulation of Segev differs to the PCSTP as he asks for a set of terminals, that *must* be included [Seg87]. The PCSTP has several applications in network design, such as e.g. designing optical backbone networks [CRR01].

Bauer and Varma considered already in 1995 the degree-constrained Steiner Tree Problem for multicasting applications in ATM networks and developed a set of heuristics to solve it [BV95]. In 1997, Jia and Wang have introduced the group multicast routing problem, in which the task is to find a multicast tree for each terminal towards all other terminals, such that the cost is minimal while *edge capacities* are not violated [JW97]. However, the authors only study an heuristic.

While CVSAP allows for arbitrary hierarchies, Aardal et al. introduced in [ACS99] the hierarchical facility location problem. In this problem, the task is to connect each terminal via a path of (open) locations such that a facility of level i can only forward its information to a node of facility $i + 1$ and so forth. Interestingy, under the assumption of stringent hierarchies, approximation algorithms are obtained by considering path decompositions.

15.3. Integer Programming Formulations

Integer Programming has successfully been applied to solve many different types of Steiner Tree Problems (see e.g. Koch et al. [KMV01]). Several different formulations have been proposed [GM93; Pol03] and the strength of these formulations is well studied [CT01; PVD01]. We note that the strength of formulations is formally defined by mappings between the corresponding solution spaces, i.e. polyhedra, and (strict) subset relations (see [BW05] for an introduction).

16. Summary of Results & Future Work

In this section, the most important results and contributions are summarized. Where applicable, possible avenues for future work are outlined.

CVSAP Definition and Model

In Section 2.2 the first concise, graph-theoretic definition for CVSAP is given. While not a contribution in its own right, we note that the previous attempt to formalize a variant of CVSAP by Oliveira and Pardalos [OP11] was inherently flawed (see [RS13b] for a discussion).

As shortly discussed in Section 15, price-collecting variants of the Steiner Tree Problem can be used to model network design problems. We note that especially the budgeted variant, where the allows costs for opening Steiner nodes is limited, may be of importance for planning the deployment of services under strict monetary restrictions. Furthermore, net worth variants may be of importance e.g. in geo-replicated services or caches: while costs are associated with installing processing functionality and using edges, by connecting terminals prizes can be collected as customers will e.g. pay for the service. We only note that this net worth variant can be easily modeled using the formulation IP-A-CVSAP by attributing negative costs (the prices) with the edges from the super source towards the terminals and dropping the constraint, that each terminal *must* receive one unit of flow.

Inapproximability Result for CVSAP

While conceptually easy to grasp based on the edge capacities, the Theorem 2.8 is instrumental for not trying to obtain approximation algorithms and instead focussing on exact or heuristic algorithms.

To complete the discussion of the computational complexity of the variants, it might be of interest to prove the inapproximability of CVSTP or NVSAP, if possible.

Approximation Algorithms for Variants

In Section 3 for three of the five CVSAP approximation algorithms are obtained. Even though the obtained approximation algorithms rely on reductions to known approximation algorithms, the found reductions might be of interest to other related optimization problems as well. For example Algorithm Leafify shows how any DNSTP solution can be converted into a DNSTP solution, in which terminals are leaves.

VirtuCast IP Formulation IP-A-CVSAP

While the formulation A-CVSAP-MCF relies on a few simple insights and models each path explicitly, the formulation IP-A-CVSAP employed in the VirtuCast algorithm relies on a compact single-commodity flow. As shown in the computational evaluation (see Sections 13 and 14.2) the VirtuCast formulation significantly outperforms the multi-commodity flow formulation and only enables solving realistically sized instances.

While in Observation 13.1 the *computational* strength of formulation IP-A-CVSAP is established, the strength of the formulations was not formally compared by considering polyhedral inclusions (cf. Section 15.3).

Even though the formal proof that IP-A-CVSAP is a stronger formulation than A-CVSAP-MCF might not be necessary based on the results of the computational evaluation, we note that especially on fat tree instances of graph sizes larger than I, the improvement of the dual bound is negligible. Since the fat tree topologies are very dense, these might indeed pose *computationally* hard instances. Nonetheless, based on the very regular structure of fat trees, it could be possible to derive a special set of cuts to improve the performance.

Algorithm Decompose

Arguably the main contribution of this thesis is the novel Algorithm Decompose, which allows to recover a feasible virtual arborescence from a solution to IP-A-CVSAP, even though the (flow) solution might contain cycles and arbitrary hierarchies. Coupled together with formulation IP-A-CVSAP, the VirtuCast algorithm is obtained.

LP-Based Heuristics

In Section 8 three different types of LP-based heuristics are presented, namely GreedyDiving, MultipleShots and FlowDecoRound. In the extensive computational evalutation in Section 12 the performance of these heuristics (with their derived variants) is evaluated. The first important established result is the high efficiency in constructing solutions (except for FlowDecoRound). Secondly, Observations 12.2 and 12.3 establish a runtime-quality trade-off accross nearly all topologies and graph sizes. Based on this trade-off, different heuristics may be chosen depending on the time available to obtain a solution. Lastly, while requiring the highest runtime, the GreedySelect heuristic *reliably* finds the best solutions *across all instances and graph sizes*.

VirtuCast Solver

By incorporating the LP-based heuristics into the VirtuCast algorithm, a solver is obtained that outperforms all other considered solution approaches (see Section 14). Using this solver, all considered instances are solved to less than 6% within optimality and a median gap of less than 1.5% can be observed on IGen and 3D torus instances.

Bibliography

[Ach07] Tobias Achterberg. “Constraint Integer Programming”. PhD thesis. TU Berlin, 2007.

[Ach09] Tobias Achterberg. “SCIP: solving constraint integer programs”. In: *Mathematical Programming Computation* 1.1 (2009), pp. 1–41.

[ACS99] Karen Aardal, Fabian A Chudak, and David B Shmoys. “A 3-approximation algorithm for the k-level uncapacitated facility location problem”. In: *Information Processing Letters* 72.5 (1999), pp. 161–167.

[AFLV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. “A scalable, commodity data center network architecture”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 38. 4. ACM. 2008, pp. 63–74.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993, pp. I–XV, 1–846. ISBN: 978-0-13-617549-0.

[And+05] Thomas Anderson et al. “Overcoming the Internet impasse through virtualization”. In: *Computer* 38.4 (2005), pp. 34–41.

[Ban+03] Suman Banerjee et al. “Construction of an efficient overlay multicast infrastructure for real-time applications”. In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 2. IEEE. 2003, pp. 1521–1531.

[BH09] Luiz André Barroso and Urs Hölzle. “The datacenter as a computer: An introduction to the design of warehouse-scale machines”. In: *Synthesis Lectures on Computer Architecture* 4.1 (2009), pp. 1–108.

[BLM13] Andreas Bley, Ivana Ljubić, and Olaf Maurer. “Lagrangian decompositions for the two-level FTTx network design problem”. In: *EURO Journal on Computational Optimization* 1.3-4 (2013), pp. 221–252.

[BV95] Fred Bauer and Anujan Varma. “Degree-constrained multicasting in point-to-point networks”. In: *INFOCOM’95. Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Bringing Information to People. Proceedings. IEEE*. IEEE. 1995, pp. 369–376.

[BW05] Dimitris Bertsimas and Robert Weismantel. *Optimization over Integers*. Dynamic Ideas, 2005.

[CB10] NM Chowdhury and Raouf Boutaba. “A survey of network virtualization”. In: *Computer Networks* 54.5 (2010), pp. 862–876.

[CCL09] Alysson M Costa, Jean-François Cordeau, and Gilbert Laporte. “Models and branch-and-cut algorithms for the Steiner tree problem with revenues, budget and hop constraints”. In: *Networks* 53.2 (2009), pp. 141–159.

[Cha+98] Moses Charikar et al. “Approximation algorithms for directed Steiner problems”. In: *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 1998, pp. 192–200.

[Cos+12] Paolo Costa et al. “Camdoop: Exploiting In-network Aggregation for Big Data Applications”. In: *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2012.

[Cra+03] Chuck Cranor et al. “Gigascope: A Stream Database for Network Applications”. In: *Proc. ACM SIGMOD International Conference on Management of Data*. 2003, pp. 647–651.

[CRR01] Suzana A Canuto, Mauricio GC Resende, and Celso C Ribeiro. “Local search with perturbations for the prize-collecting Steiner tree problem in graphs”. In: *Networks* 38.1 (2001), pp. 50–58.

[CT01] Sunil Chopra and Chih-Yang Tsai. “Polyhedral approaches for the Steiner tree problem on graphs”. In: *COMBINATORIAL OPTIMIZATION-DORDRECHT-* 11 (2001), pp. 175–202.

[DCX03] Min Ding, Xiuzhen Cheng, and Guoliang Xue. “Aggregation tree construction in sensor networks”. In: *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*. Vol. 4. IEEE. 2003, pp. 2168–2172.

[Eis+10] Friedrich Eisenbrand et al. “Connected facility location via random facility sampling and core detouring”. In: *Journal of Computer and System Sciences* 76.8 (2010), pp. 709–726.

[Fuc03] Bernhard Fuchs. “A note on the terminal Steiner tree problem”. In: *Information Processing Letters* 87.4 (2003), pp. 219–220.

[GLS] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization. 1988*.

[GM93] Michel X Goemans and Young-Soo Myung. “A catalog of Steiner tree formulations”. In: *Networks* 23.1 (1993), pp. 19–28.

[Her+07] Christian Hermsmeyer et al. “Ethernet aggregation and core network models for effcient and reliable IPTV services”. In: *Bell Labs Technical Journal* 12.1 (2007), pp. 57–76. ISSN: 1538-7305.

[HHR13] Ludovic Henrio, Fabrice Huet, and Justine Rochas. “An Optimal Broadcast Algorithm for Content-Addressable Networks”. In: *Principles of Distributed Systems*. Springer, 2013, pp. 176–190.

[Hu+04] X-D Hu et al. “Multicast routing and wavelength assignment in WDM networks with limited drop-offs”. In: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. Vol. 1. IEEE. 2004.

[JMP00] David S. Johnson, Maria Minkoff, and Steven Phillips. “The prize collecting Steiner tree problem: theory and practice”. In: *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. SODA ’00. San Francisco, California, USA, 2000, pp. 760–769.

[JW97] Xiaohua Jia and Lusheng Wang. “A group multicast routing algorithm by using multiple minimum Steiner trees”. In: *Computer communications* 20.9 (1997), pp. 750–758.

[KEW02] Bhaskar Krishnamachari, Deborah Estrin, and Stephen Wicker. “Modelling data-centric routing in wireless sensor networks”. In: *IEEE infocom*. Vol. 2. 2002, pp. 39–44.

[KM98] Thorsten Koch and Alexander Martin. “Solving Steiner tree problems in graphs to optimality”. In: *Networks* 32.3 (1998), pp. 207–232.

[KMV01] Thorsten Koch, Alexander Martin, and Stefan Voß. “SteinLib: An updated library on Steiner tree problems in graphs”. In: *COMBINATORIAL OPTIMIZATION-DORDRECHT-* 11 (2001), pp. 285–326.

[KV12] B Bernhard H Korte and Jens Vygen. *Combinatorial optimization*. Springer, 2012.

[KVM02] Fernando Kuipers and Piet Van Mieghem. “MAMCRA: a constrained-based multicast routing algorithm”. In: *Computer Communications* 25.8 (2002), pp. 802–811.

[LR04] Abilio Lucena and Mauricio GC Resende. “Strong lower bounds for the prize collecting Steiner problem in graphs”. In: *Discrete Applied Mathematics* 141.1 (2004), pp. 277–294.

[MG07] Jiří Matoušek and Bernd Gärtner. *Understanding and using linear programming*. Springer, 2007.

[Nar+13] Srinivas Narayana et al. “Joint Server Selection and Routing for Geo-Replicated Services”. In: *Proc. Workshop on Distributed Cloud Computing (DCC)*. 2013.

[OP11] CarlosA.S. Oliveira and PanosM. Pardalos. “Streaming Cache Placement”. In: *Mathematical Aspects of Network Routing Optimization*. Springer Optimization and Its Applications. Springer New York, 2011, pp. 117–133. ISBN: 978-1-4614-0310-4.

[PLK10] Ju-Won Park, Huhnkuk Lim, and JongWon Kim. “Virtual-node-based multicast routing and wavelength assignment in sparse-splitting optical networks”. In: *Photonic Network Communications* 19.2 (2010), pp. 182–191.

[Pol03] Tobias Polzin. “Algorithms for the Steiner problem in networks”. PhD thesis. Universitätsbibliothek, 2003.

[PVD01] Tobias Polzin and Siavash Vahdati Daneshmand. “A comparison of Steiner tree relaxations”. In: *Discrete Applied Mathematics* 112.1 (2001), pp. 241–261.

[Quo+09] B. Quoitin et al. “IGen: Generation of router-level Internet topologies through network design heuristics”. In: *Proc. 21st International Teletraffic Congress (ITC)*. 2009, pp. 1–8.

[Rav+01] R Ravi et al. “Approximation algorithms for degree-constrained minimum-cost network-design problems”. In: *Algorithmica* 31.1 (2001), pp. 58–78.

[Rou03] George N Rouskas. “Optical layer multicast: rationale, building blocks, and challenges”. In: *Network, IEEE* 17.1 (2003), pp. 60–65.

[RS13a] Matthias Rost and Stefan Schmid. *CVSAP-Project Website*. http://www.net.t-labs.tu-berlin.de/~stefan/cvsap.html, 2013.

[RS13b] Matthias Rost and Stefan Schmid. *The Constrained Virtual Steiner Arborescence Problem: Formal Definition, Single-Commodity Integer Programming Formulation and Computational Evaluation*. Tech. rep. arXiv, 2013.

[RS13c] Matthias Rost and Stefan Schmid. “VirtuCast: Multicast and Aggregation with In-Network Processing”. In: *Principles of Distributed Systems*. Ed. by Roberto Baldoni, Nicolas Nisse, and Maarten Steen. Vol. 8304. Lecture Notes in Computer Science. Springer International Publishing, 2013, pp. 221–235.

[Sch98] Alexander Schrijver. *Theory of linear and integer programming*. Wiley, 1998.

[Seg87] Arie Segev. “The node-weighted steiner tree problem”. In: *Networks* 17.1 (1987), pp. 1–17.

[Shi01] Sherlia Shi. “A Proposal for A Scalable Internet Multicast Architecture”. In: *Washington Universtiy*. 2001.

[Voß06] Stefan Voß. “Handbook of optimization in telecommunications”. In: ed. by Mauricio GC Resende and Panos M Pardalos. Spinger Science + Business Media, New York, 2006. Chap. 18.

[WS13] Yang Wang and Wei Shi. “On Scheduling Algorithms for MapReduce Jobs in Heterogeneous Clouds with Budget Constraints”. In: *Principles of Distributed Systems*. Springer, 2013, pp. 251–265.

[CPL13] CPLEX. http://www.cplex.com/. 2013.

[Eur12] European Telecommunications Standards Institute. “Network Functions Virtualisation - Introductory White Paper”. In: *SDN and OpenFlow World Congress, Darmstadt-Germany* (2012).

[GNU13] GNU Linear Programming Kit. http://www.gnu.org/software/glpk/glpk.html. 2013.