Routing, Scheduling, and Sorting in Consolidated Networks

by

Madison Van Dyk

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2024

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Michael Hewitt, Professor Information Systems and Supply Chain Management Department Loyola University Chicago
Supervisor:	Jochen Koenemann, Professor Department of Combinatorics and Optimization University of Waterloo
Internal Members:	Joseph Cheriyan, Professor Department of Combinatorics and Optimization University of Waterloo
	Ricardo Fukasawa, Professor Department of Combinatorics and Optimization University of Waterloo
	Chaitanya Swamy, Professor Department of Combinatorics and Optimization University of Waterloo
Internal-External Member:	Sibel Alumur Alev, Associate Professor Department of Management Science and Engineering University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The results in this thesis are based on joint work with my supervisor Jochen Koenemann. I was responsible for drafting the results, and received help and feedback for revisions.

The main results of Chapters 3, 4, and 6 of this thesis are based on the following papers that I have coauthored, respectively.

1. [61]: *Sparse dynamic discretization discovery via arc-dependent time discretizations.* Joint work with Jochen Koenemann. https://arxiv.org/abs/2305.19176.
2. [60]: *Dynamic discretization discovery under hard node storage constraints.* Joint work with Jochen Koenemann. https://arxiv.org/abs/2303.01419.
3. [59]: *Fast combinatorial algorithms for efficient sortation.* Joint work with Kim Klause, Jochen Koenemann, and Nicole Megow. To appear in Proceedings of the 25th Conference on Integer Programming and Combinatorial Optimization (IPCO 2024). https://arxiv.org/abs/2311.05094.

Abstract

Modern parcel logistic networks are designed to ship demand between given origin, destination pairs of nodes in an underlying directed network. Efficiency dictates that volume needs to be consolidated at intermediate nodes in typical hub-and-spoke fashion. In practice, such consolidation requires tracking packages in both space and time (temporal network design), as well as parcel sortation.

In the first half of the thesis, we study solution methods for temporal problems arising in consolidated networks. While many time-dependent network design problems can be formulated as time-indexed formulations, the size of these formulations depends on the discretization of the time horizon and can become prohibitively large. The recently-developed dynamic discretization discovery (DDD) method allows many time-dependent problems to become more tractable by iteratively solving instances of the problem on smaller networks where each node has its own discrete set of departure times.

There are two design elements of existing DDD algorithms that make it problematic for use in region-based hub-and-spoke networks. First, in each iteration, all arcs departing a common node share the same set of departure times. This causes DDD to be ineffective for solving problems where all near-optimal solutions require many distinct departure times at the majority of the high-degree nodes in the network, an aspect characteristic of region-based networks. A second challenge is handling static storage constraints without leading to a weak relaxation in each iteration.

To mitigate these limitations, an arc-based DDD framework is proposed in Chapter 3, where departure times are determined at the arc level instead of the node level. We apply this arc-based DDD method to instances of the service network design problem (SND). We show that an arc-based approach is particularly advantageous when instances arise from region-based networks, and when candidate paths are fixed in the base graph for each commodity. Moreover, our algorithm builds upon the existing DDD framework and achieves these improvements with only benign modifications to the original implementation. Additionally, Chapter 4 introduces bounds on additional storage required in each iteration, expanding the applicability of DDD to problems with bounded node storage, such as the universal packet routing problem. Our arguments rely solely on the structure of the standard map, μ, from the original formulation to the smaller relaxed formulations.

In order to maintain consistent operations, some models stipulate that the implemented transportation schedule must be repeated each day. In Chapter 5 we present a DDD model for solving a version of SND with cyclic constraints. We show that these cyclic constraints require new conditions on the time discretization at each node, leading to larger partial

networks. We then highlight challenges in reducing the size of partial networks as they grow over each iteration of DDD. We demonstrate that certain policies for removing departure times in each iteration can cause the iterations in DDD to repeat, preventing termination.

In the second half of this thesis, we study *parcel sortation*, an aspect of routing that has previously been left unaddressed from a theory perspective. Warehouses have limited *sort points*, the physical devices tasked with handling packages destined for a particular downstream warehouse. We propose a mathematical model for the physical requirements, and limitations of parcel sortation. We then show that it is NP-hard to determine whether a feasible sortation plan exists. We consider two natural objectives: minimizing the maximum number of sort points required at a warehouse, and minimizing the total number of sort points required in the network.

In Chapter 6, we consider the problem of minimizing the maximum number of sort points required at a warehouse. We discuss several settings, where (near-)optimality of a given sortation instance can be determined efficiently. The algorithms we propose are fast and build on combinatorial *witness set* type lower bounds that are reminiscent and extend those used in earlier work on degree-bounded spanning trees and arborescences. In Chapter 7, we present algorithms for minimizing the total number of sort points required in a network. In contrast to the min-degree setting, it is not known if this min-cardinality setting is NP-hard. In progress towards answering this question, we present fast combinatorial algorithms for solving in-tree, out-tree, and spider instances. Our algorithms are based on reduction, decomposition, and uncrossing techniques that simplify instances.

Acknowledgements

I want to express my heartfelt thanks to my advisor, Jochen Koenemann, for his guidance and support. His mentorship throughout my graduate studies has been invaluable. Our discussions over the past years have been helpful both in regard to research and beyond academia.

I am grateful to my thesis committee members, Joseph Cheriyan, Ricardo Fukasawa, Chaitanya Swamy, Michael Hewitt, and Sibel Alumur Alev for their expertise and constructive feedback.

I am fortunate to have had enjoyable collaborations in a variety of projects. Thank you to Joseph Cheriyan, Gabriel Morete, Sharat Ibrahimpur, Nicole Megow, Kim Klause, and Nikhil Kumar for all the insightful discussions. I am also grateful for the mentorship and support I received along the way from David Jao and Shai Ben-David. Both David and Shai were incredibly generous with their time, and always interested in a good research discussion. Their encouragement was greatly appreciated, and something I wish to emulate.

Thank you to the Modeling and Optimization Team at Amazon for exposure to the interesting problems motivating this thesis, as well as providing an environment with creative and encouraging people.

To the entire C&O department throughout my Undergraduate and Graduate degrees, as well as the honorary members, thank you. The intramural teams, seminars, and casual gatherings greatly contributed to my experience at Waterloo, and I will miss you. Thank you to Cedric, Sharat, Akshay, and Justin for exposing me to the world of combinatorial optimization when we shared an office during a term in undergrad ... and thank you for not informing the admins when I secretly kept my desk in that office for the subsequent term. Melissa and Carol – you are the best!

Thank you to Adam, Priya, Sharat, and Fernanda for our many discussions – academic and otherwise. Finally, thank you to my family for always supporting me, and for taking this journey with me.

I am grateful to have had the financial support of an NSERC Postgraduate Scholarship, an Ontario Graduate Scholarship, a Cotton Women in Mathematics Scholarship, and an Amazon Post-internship Fellowship.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern parcel logistic networks are designed to ship demand between given origin, destination pairs of nodes in an underlying directed network. This physical (flat) network can be expressed as a directed graph $D = (N, A)$ with the node set N representing the warehouse locations, and the arc set A representing the available transportation legs between locations. In less-than-truckload (LTL) transportation, shipments are small in size relative to the capacity of the trailers. As a result, transportation costs incentivize the consolidation of shipments onto shared trailers.

1.1 Consolidation in logistics

In order for two packages to be consolidated at a warehouse, they must be able to depart the warehouse at the same time, on the same leg. For instance, consider Figure 1.1, and suppose there is 1/3 truck load of packages that must shipped from s_i to t_j for each $i \in \{1, 2\}, j \in \{1, 2, 3\}$. Rather than send a truck directly from s_i to t_j for each pair, we instead consolidate the packages at node u, as shown in Figure 1.1. Five trucks will be required to ship the packages, provided that for each arc, all packages using that arc are scheduled to traverse it at the same time.

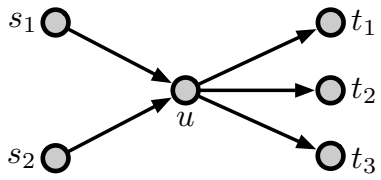


Figure 1.1

The flat network D is often designed in a way that permits opportunities for consolidation. For instance, many low-cost and efficient networks follow a region-based structure. A *hub-*

and-spoke network is a common network structure used in airline and fulfillment networks including the distribution networks of FedEx and UPS [4, 32]. In a hub-and-spoke network, locations are divided into regions and each region has a designated hub. Shipments that have origins and destinations in two different regions must travel between regions via the hubs, resulting in high-degree nodes that have significant throughput. In Figure 1.2, hubs are indicated as gray squares and only arcs for inter-region shipments are shown.

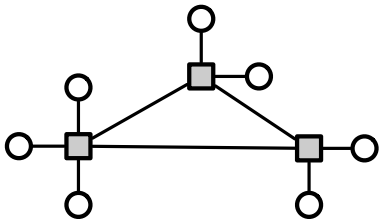


Figure 1.2: Hub-and-spoke network

To determine package routes and schedules that take advantage of consolidation opportunities to minimize shipment costs, package flows must be modelled in both space and time. These considerations are captured by temporal network design, a class of problems involving the optimization and planning of network infrastructure that operates dynamically over time. Unlike traditional network design problems that focus on static configurations, temporal network design considers the time-varying aspects of network behavior, such as arc transit times, and shipment release times and deadlines. Temporal network design problems are often modelled using *time-indexed* formulations which have variables and constraints indexed by each time point a decision could be made [2]. However, as the time discretization becomes fine, the size of the corresponding formulation becomes unreasonably large and cannot be solved by even state-of-the-art solvers in a reasonable amount of time. Significant work has focused on improving solution methods for these problems [2, 18, 63].

In addition to the problem of routing and scheduling shipments, the "under-the-roof" processes in a warehouse capture the need for parcel sortation in networks with consolidation. Each package is labelled with its origin and destination, as well as its assigned route through the network. In the simplest model, at each warehouse, packages are unloaded from inbound trailers, and sorted and loaded onto outbound trailers based on their assigned route. To improve efficiency, packages are grouped into containers, which are in turn loaded onto trailers. Each container is labelled with the warehouse at which it is loaded, the warehouse at which it is unloaded, and its assigned path between these two warehouses. At each intermediate warehouse along this path, rather than unloading and

sorting the packages in the container, the container bypasses the sortation process and is loaded directly onto an outbound trailer. This process is depicted in Figure 1.3.

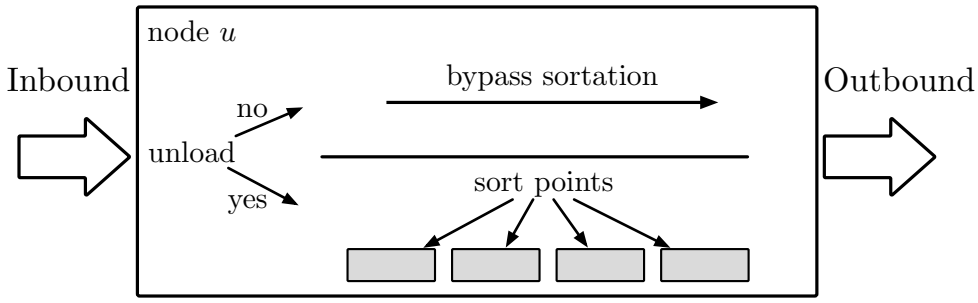


Figure 1.3: Sample intra-warehouse procedure.

When packages are unloaded at a warehouse, they are sorted, based on the package label, into containers. We refer to the physical device tasked with packages destined for one specific downstream node as a *sort point*, shown in Figure 1.3. A natural problem that arises is to determine a sort point assignment throughout the fulfillment network in order to ship all packages, given limited sort point space at each warehouse. This problem and related questions have thus far been left unaddressed from a theory perspective.

This thesis is split into two parts. The first part focuses on temporal network design and solution methods for problems arising in consolidated networks. We focus specifically on the recent algorithmic approach of *dynamic discretization discovery (DDD)*, introduced by Boland, Hewitt, Marshall, and Savelsbergh [2]. The DDD framework was first applied to the *service network design (SND)* problem, a fixed-charge problem that is frequently used to model the coordination of the physical route of the shipments as well as the departure time for each leg of the journey. We present a new arc-based DDD approach that results in smaller iterations for many graph structures, including hub-and-spoke networks, allowing the algorithm to terminate more quickly. We also present modelling enhancements that broaden the application of DDD to problems with static node storage and cyclic constraints. Background on DDD and an overview of our contributions is presented in Sections 1.2 and 1.3.

The second part of this thesis considers parcel sortation. We present an abstract model for sortation and introduce two natural objectives to consider when allocating sort points. We prove that given a fixed sort point capacity at each warehouse, determining whether or not there is a feasible sort point allocation is NP-hard. We then consider particular instance classes and present exact and approximation algorithms. Our model for sortation, along with an overview of our contributions, is provided in Sections 1.4 and 1.5. The goal

underpinning the work in both Parts I and II is to improve theoretical understanding and algorithmic approaches for problems arising in networks with consolidation.

1.2 Temporal network design and DDD

The first focus of this thesis is the advancement of the DDD framework, a paradigm which applies to a broad class of temporal network design problems. In this section we provide an introduction to time-indexed formulations for temporal network design and the DDD paradigm. A more detailed and technical introduction is given in Chapter 2.

1.2.1 Temporal network design

There are two main model types used to solve temporal network design problems: continuous formulations and time-indexed formulations. *Continuous formulations* have a continuous variable for each commodity and arc pair, and its value is equal to the time the commodity traverses the arc. Practical temporal problems are solved over some specified time horizon T with an associated time discretization $[T] := \{0, 1, \dots, T\}$ indicating the times at which decisions can be made (ex. the times at which trucks can depart warehouses). *Time-indexed formulations* have variables and constraints indexed by each time point in $[T]$ [57].

Time-indexed formulations were first introduced by Ford and Fulkerson [19, 20] in the context of network flow theory. The authors demonstrated that these problems can be reframed as static network flow problems in the corresponding *(fully) time-expanded network*, D_T, which has copies of each node and arc in D for each time $t \in [T]$. We refer to nodes and arcs in D_T as *timed nodes* and *timed arcs* respectively. While continuous formulations are have fewer variables and constraints than time-indexed formulations, they require "big-M" constraints in order to enforce binary decisions such as fixed costs. As a result, these formulations often have slow solve times due to large branch-and-bound trees [39]. Time-indexed formulations have stronger linear relaxations, but these formulations become impractical to solve for fine time discretizations since the number of variables (and size of D_T) grows linearly in T [19, 20]. Moreover, many natural multicommodity flows problems are NP-hard in the temporal setting [27].

Since these time-indexed formulations are often impractical to solve directly with a mixed-integer programming solver, significant work has focused on alternative methods for achieving exact and approximate solutions. Fleischer et al. [17, 18] provide guarantees on the cost increase of the optimal solution when we allow a coarser network and only include

node copies for every Δ units of time. In this Δ-condensed approach, each node shares the same set of departure times $\{0, \Delta, 2\Delta, \ldots, T\}$, as opposed to a partially time-expanded network in DDD, where the departure times available to each node is non-uniform.

In a similar vein of working with a reduced time discretization, Wang and Regan [64] show that iteratively refining a time window discretization for the travelling salesman problem with time windows will converge to an optimal solution. Similarly, Dash et al. [13] iteratively refine a set of time periods based on a preprocessing scheme. A key improvement of the DDD approach of Boland et al. over the previous iterative approaches is that it refines a time discretization by using information from solutions to the lower bound formulations [2].

1.2.2 Dynamic Discretization Discovery

While an optimal solution may require a large number of departure times in the network, it may be the case that each individual node only requires a small number of departure times. The recently developed *dynamic discretization discovery* (DDD) method of Boland et al. [2] allows many time-dependent (minimization)[1] problems to become more tractable by taking advantage of this observation. A DDD algorithm solves a series of mixed-integer programs (MIPs) defined on smaller *partially time-expanded networks*, denoted D_S, that include only a subset of the timed nodes in the fully time-expanded network along with shortened timed arcs. The partially time-expanded networks and corresponding MIPs are constructed to ensure that they provide a lower bound on the optimal value of the original problem. If a solution to the MIP defined on the partially time-expanded network can be converted to a solution to the original instance of equal cost, then an optimal solution has been found. Otherwise, the partially time-expanded network is refined by adding timed nodes and timed arcs based on the current solution, and the process is repeated. In effect, each node begins with a coarse time discretization which only permits a small subset of departure times in $[T]$. In each iteration the time discretization is refined at each node non-uniformly until an optimal solution is found. When a time discretization is refined at a node, departure times from $[T]$ are added to the existing set of permissible departure times from that node. Thus, the DDD framework has the potential to determine which time points are needed at each node in an optimal solution, without ever constructing the fully time-expanded network.

There are two design elements of existing DDD algorithms that make it problematic for

[1]The DDD framework applies to both minimization and maximization problems. Without loss of generality we will consider minimization problems in this thesis.

use in region-based hub-and-spoke networks. First, in each iteration, all arcs departing a common node share the same set of departure times. This causes DDD to be ineffective for solving problems where all near-optimal solutions require many distinct departure times at the majority of the high-degree nodes in the network. Region-based networks are one such structure that often leads to many high-degree nodes, and their increasing popularity underscores the importance of tailoring solution methods for these networks. A second challenge is handling static storage constraints without leading to a weak relaxation in each iteration. The process of consolidation necessitates the ability to store packages and carts in a warehouse, since packages must remain in a warehouse until an appropriate outbound truck is scheduled for departure.

1.3 Contributions for DDD

In this thesis we address each of these challenges. We develop a new arc-based DDD approach that results in smaller formulations for many graph structures. We also prove bounds on the additional storage that must be permitted at each timed node, which generalizes the application of DDD to problems with static node capacity constraints. A description of each of these contributions is given in Sections 1.3.1 and 1.3.2.

In order to maintain consistent operations, some models stipulate that the implemented transportation schedule must be repeated each day. We present a DDD model for solving a version of SND with cyclic constraints used to enforce this assumption. We show that these cyclic constraints require new conditions on the time discretization at each node, leading to larger partial networks. We then highlight challenges in reducing the size of partial networks as they grow over each iteration of DDD. A recently-proposed strategy for such a reduction was to remove timed nodes when they are no longer required in a high-quality solution [54]. We present a concrete roadblock in this approach, by showing that certain policies for removing timed nodes from the current partial network can prevent a DDD algorithm from terminating. These contributions are described in Section 1.3.3.

1.3.1 Arc-based DDD

For many problems, while each node may require a large number of departure times in a near-optimal solution, each *arc* may only require a small number of departure times. For these problems, an optimal solution can be expressed on a time-expanded network with a small number of timed arcs relative to the arc set in the full time-expanded network. With this sparsity in mind, we enhance the DDD paradigm by developing an approach where the set of departure times are determined on the arc level rather than the node level.

Our arc-based DDD approach broadens the application of the DDD paradigm to problems that require a large number of departure times at each node but not each arc. We apply this arc-based approach to the problem of SND-RR, which is a variant of service network design where each commodity has a designated feasible network in space through which it must be routed. We show that this arc-based approach offers speed-up for region-based networks following a hub-and-spoke structure. We also demonstrate that our arc-based DDD algorithm offers improvement over the node-based approach when applied to problems where shipment paths are fixed in the flat (physical) network, as was the case in [40]. Moreover, our algorithm builds upon the existing DDD framework and achieves these improvements with only simple modifications to the original implementation.

Our main contributions are as follows:

1. We develop an arc-based DDD approach that decreases the size of the formulation solved in each iteration for many network structures.
2. We prove that the arc-based DDD approach generalizes the node-based approach (Theorem 3.6). Specifically, given any initial partial network and refinement process defining a node-based DDD algorithm, there is a corresponding initial partial (auxiliary) network and refinement process for the arc-based DDD approach which leads to the same algorithm.
3. We implement an arc-based DDD algorithm for the problem of SND-RR and apply the algorithm to families of instances based on the original construction outlined in [2]. We consider three sets of instances:
   (a) SND-RR with designated paths: each commodity has a designated path in the flat (physical) network;
   (b) SND on hub-and-spoke networks: the flat network has a hub-and-spoke structure and commodity routes are unrestricted;
   (c) SND with critical times: the flat network is randomly chosen as in [2], commodity routes are unrestricted, and each node has a set of critical times (release times and deadlines).

   Our computational results are summarized in Table 1.1, where the variable and constraint counts are with respect to the lower bound formulation in the final iteration of each DDD algorithm. Each entry indicates the average percentage reduction in that field when solving an instance with the arc-based DDD approach compared to solving the instance with the node-based DDD approach.

Family of instances	runtime	variable count	constraint count
SND-RR with designated paths	57%	45%	40%
SND on hub-and-spoke networks	42%	34%	31%
SND with critical times	12%	20%	19%

Table 1.1: Average % decrease for arc-based DDD over node-based DDD.

The implementation of the arc-based and node-based DDD approaches as well as the benchmark instances can be found at https://github.com/madisonvandyk/SND-RR.

1.3.2 Node storage

Many of the original applications of DDD were for problems with no storage capacity constraints at nodes. In turn, the proofs for the correctness of the lower bounds rely on the freedom of unlimited storage so that flow arriving "early" does not lead to infeasibility. This need for unlimited node storage has previously prevented the DDD model from addressing many real-world problems, such the dynamic scheduling problem considered by Lara et al. for which only heuristic techniques are known for large instances [40]. Removing DDD's reliance on unbounded zero-cost storage was noted as an important direction of future research by Boland and Savelsbergh [3]. In Section 1.3.4 we provide an overview of related work that has made progress in this direction.

While DDD is often applied to continuous problems without a prespecified discretization, many practical routing problems are in fact discrete in nature, such as the dynamic scheduling problem considered by Lara et al. [40]. Hewitt and Crainic [12] also state that in many applications, continuous problems are modelled as discrete problems the time granularity is chosen based on the application at hand. With this in mind, we address the relaxation of static node storage constraints for problems with discrete time discretizations. We prove upper bounds on the storage that must be permitted at each timed node based on the current time-expanded network and the arc and storage capacities in the underlying network. Our arguments rely only on the standard map μ from timed arcs in the fully time-expanded network to timed arcs in the partially time-expanded network. As a result, they generalize to fixed-charge problems, including the service network design problem.

We demonstrate our techniques in the case of the classical *universal packet routing* (UPR) problem. In an instance of UPR, we are given a directed graph where each arc has a transit time and throughput capacity, and each node has a storage capacity. Given a set of source-sink pairs (packets), the objective of UPR is to route all packets through the network in order to minimize the makespan of the schedule, while respecting arc capacities

and node storage levels at every point in time. This problem is NP-hard, and even the special case where node capacities are zero is as hard to approximate as vertex colouring [5]. Focusing on UPR allows us to clearly present the extension of the DDD method to a problem with bounded node storage without complicating constraints already addressed by previous work.

The contributions in this work are both algorithmic and theoretical. The main ingredients of our contributions are as follows:

1. We develop and implement a lower bound model and refinement process for time-indexed problems with bounded node storage;
2. To prove that the lower bound model is in fact a relaxation, we present arguments relying on structural observations of the map from the fully time-expanded network to the partially time-expanded network;
3. We prove that with our lower bound and refinement process, the algorithm terminates with an optimal solution in at most $|N|T^*$ iterations, where N is the set of nodes in the base graph and T^* is the minimum makespan;
4. We implement and test our DDD algorithm on two classes of instances: one based on the population centres of the United States, and the other based on social networks. We demonstrate that our DDD algorithm completes in an average of 53% of the time of solving the full time-indexed formulation when the known upper bound is $2T^*$ for a class of geographic instances, and 49% of the time for a class of geometric instances.

1.3.3 Additional DDD contributions

In many practical networks, commodity demands and network operations often have a repetitive structure. That is, the origin-destination flow and corresponding trajectories of packages released on day one are roughly repeated each subsequent day. As a result, it is often desirable to implement routing operations that repeat each day since they are consistent and easier to manage. While in reality day-to-day operations are not always exactly the same, such an assumption is at times made in the planning stage of network design [6, 40]. With these assumptions, the problem of computing the optimal operation of a network simplifies to finding an optimal *daily* operation, while taking into consideration the flow released over the course of the whole time horizon.

To compute an optimal daily network operation while incorporating all flow throughout the time horizon, Lara et al. [40] incorporate *cyclic capacity constraints*. For each arc $a \in A$ and hour $t \in [24]$, a cyclic constraint for arc a at time t combines flows departing arc a at time t during *any* day. In the implementation of this constraint, we use the modulo

operation. In Chapter 5 we present a DDD algorithm for SND with cyclic constraints. We introduce a new condition on the set of timed nodes that ensures an optimal routing through the partial network provides a lower bound. We observe that cyclic constraints lead to larger sets of timed nodes required in partial networks.

Since the set of timed nodes in the partial networks increases significantly in each iteration, we investigate the proposed strategy of Scherr et al. [54] to remove timed nodes and timed arcs from the current partial network if they are no longer required for obtaining high-quality solutions. An initial strategy for removing timed nodes would be to remove any timed node that is not used in the current solution to the lower bound model defined on the partial network, provided that properties of the lower bound model can still be satisfied with the remaining timed nodes. However, we prove that this strategy can lead iterations to repeat, preventing the DDD algorithm from terminating.

Our contributions are summarized as follows:

1. We define conditions on the timed node set that ensure an optimal routing through the partial network provides a lower bound given cyclic capacity constraints;
2. We provide a DDD framework that can be applied to SND with cyclic constraints;
3. We show that the policy of removing timed nodes when they are not used in the current feasible solution can cause iterations to cycle.

1.3.4 Related work

We now provide an overview of related work on DDD and UPR.

Dynamic discretization discovery

Initial applications of the DDD framework addressed connectivity problems such as the shortest path problem [28] and the travelling salesman problem with time windows [62]. In the case of SND [2] where trucks have capacities, there is no bound on the number of trucks that can travel along a specified arc at any given time. The same assumption is made by Scherr et al. [54] and Hewitt [31] when applying DDD to variants of SND. These assumptions avoid complicating capacity constraints that can be problematic when mapping a solution from the fully time-expanded network to the partially time-expanded network, and vice versa. For a complete presentation of the DDD framework for connectivity problems, we refer the reader to the survey of Boland and Savelsbergh [3].

More recently, DDD has been applied to problems with warehouse capacity constraints and costs. He et al. [29] consider a variant of SND where each warehouse has a given

loading capacity and unloading capacity. The authors assume that trucks can traverse a transportation leg slower than its stated transit time without incurring any penalty. As a result, the partial networks they use differ from the standard construction. Additionally, this variant of warehouse capacity can be modelled as an arc capacity constraint by working with an auxiliary graph that has an "unload" copy and a "load" copy for each warehouse. Similarly, in the scheduling problem with time-dependent durations and resource consumptions constraints considered by Pottel and Goel [50], the resource constraints at warehouses can be encoded using arc capacities.

Lagos et al. [39] consider the *continuous inventory routing problem* (CIR), in which a company manages the inventory of its clients, and delivers product from a single facility. The authors also encode two different storage capacity constraints. While each client has a storage limit, the authors assume that products that arrive at a client location do not impact the storage level unless a delivery is scheduled. In many problems, including UPR and SND, all stationary flow must count towards the storage level at some node. This aligns more closely with the constraint of Lagos et al. [39] that at most one vehicle can visit a fixed client at any point in time. The results in [39] do not extend to the node storage constraints considered in this thesis since they rely on structural results unique to CIR.

Shu et al. [55] apply DDD to the problem of SND where each node has a per-unit-of-demand-and-time holding cost. The relaxation of holding costs differs from the relaxation required for the static node storage constraints of UPR. In the case of holding costs, the relaxed storage cost incurred is computed independently for each commodity. In contrast, the relaxed node storage capacity must consider all commodities for each timed node. This treatment of all commodities simultaneously is also central to the tightened constraints presented in Section 4.2.2 (See Lemma 4.5).

Strategies to improve the DDD algorithm focus on limiting the number of iterations until termination, as well as speeding up each individual iteration. Marshall et al. [45] introduce the *interval-based dynamic discretization discovery algorithm* (DDDI) which was demonstrated to find solutions to instances of SND faster than traditional DDD. In DDDI, the time discretization in each iteration is interpreted as a set of time intervals at each node, rather than the set of departure times at each node. This interpretation allows for a more effective refinement strategy so that in each iteration, the current optimal flat solution in the partial network (set of physical paths and consolidation of commodities onto arcs) is infeasible in the subsequent iteration. Scherr et al. [54] suggest removing timed nodes if they are no longer required for a high quality solution. However, no removal strategy has been implemented or studied in the literature. Hewitt [30] explores speed-up techniques

for DDD which include enhancements such as a two-phase implementation of DDD and the addition of cutting planes to strengthen the relaxed model in each iteration.

Despite noteworthy improvements to the DDD paradigm, all of the previously proposed algorithm enhancements maintain the structure that all arcs departing a common node share the same set of departure times in each iteration. This is true even in the case of DDDI, where all arcs departing a common node share the same set of time intervals. While in DDDI each commodity has its own partial network, these networks are constructed by removing timed arcs that could not be used by that commodity in *any* solution. However, the issue of large partial networks remains in DDDI when most timed arcs could be used in a feasible solution, and thus cannot be removed. As a result, even with these enhancements, the DDD framework is ineffective for problems where all near-optimal solutions require a large number of departure times at the majority of the nodes in the network. For this reason, our work tackles a previously unaddressed issue. Since our arc-based approach is orthogonal to the previously proposed improvements to DDD, our experimental results compare to the original DDD implementation for SND presented in [2]. We note that the same arc-based approach naturally extends to the setting of DDDI.

Packet routing

Packet routing in the literature refers to a broad range of problems, closely related to our definition of UPR in this paper. In *store-and-forward packet routing* (SF-PR), arcs can only accommodate a single packet at any given time, and transit times are unit length. Additionally, each packet has a specified path it must follow in the underlying network. The *congestion* C denotes the maximal number of paths using a single arc in the base graph, and the *dilation* D denotes the maximal length of a path along which a packet must be routed. An $O(C + D)$ approximation for SF-PR was originally developed by Leighton et al. [43, 44]. Building upon these initial results, various papers have established polytime approximation results for SF-PR with arbitrary arc capacities and transit times [49, 53]. However, the constants in these guarantees remain large for general graphs [49] ($39(C+D)$ for general graphs, and $23.4(C + D)$ when there are unit transit times and arc capacities). SF-PR was proven to be NP-hard by Di Ianni [15]. Peis et al. [48, 49] generalized SF-PR to include arbitrary arc capacities and transit times and proved that this problem is APX-hard. We note that our UPR problem is more general since we add node storage constraints. In this paper, we refer to the variant of UPR where each packet is given a designated path in the base network as the problem of *universal packet routing with fixed paths* (UPR-FP).

Current approximation strategies for packet routing problems (with variable paths) involve

converting the problem to an instance of UPR-FP by selecting an appropriate path for each packet. Busch et al. [5] consider the *bufferless* packet routing problem, in which once a packet is injected into the network, it cannot be stored at any node. The authors prove that this problem is not only NP-hard, but as hard to approximate as vertex colouring. Vertex colouring is hard to approximate within $\Omega(n^{1-\epsilon})$ for any $\epsilon > 0$, assuming NP $\subsetneq$ ZPP, where n denotes the number of vertices in the graph [16]. In order to ensure that a feasible solution to an instance of UPR is indeed contained in some time-indexed formulation, we must allow time-indexed variables for times up to some known upper bound on the min makespan, T^*. By our previous discussion, it is not reasonable to expect that we know an upper bound T on T^* where $T \approx T^*$. This motivates the importance of applying DDD to solve this problem, since we find that the upper bound provided has much less of an impact on the runtime of DDD compared to solving the full time-indexed formulation.

1.4 Sortation

In Part II we examine a model for allocating sort points in a fulfillment network. As input, we are given a fulfillment network represented as a directed graph $D = (N, A)$, and a collection of commodities $\{(s_k, t_k)\}_{k \in \mathcal{K}}$, consisting of pairs of source and sink nodes in N. Furthermore, each commodity k comes with a directed s_k, t_k-path, P_k, along which all packages associated with commodity k must travel. This assumption is often made in practical applications, as discussed in [40]. Note that while o_k and d_k are standard for representing the source and sink in temporal network design, we use s_k and t_k in our work on sortation since these are standard for connectivity problems. Let n and m represent the number of nodes and arcs in D, respectively.

In the simplest setting, each package that travels from s_k to t_k via some internal node u must be *sorted* at u for its subsequent downstream node. In Figure 1.4, packages arrive at node u from nodes $s_1, \dots s_i$ and travel on to downstream nodes $v_1, \dots, v_j$. This requires sortation at node u to sub-divide the stream of incoming parcels between the j possible next stops. We refer to the physical device tasked with packages destined for one specific downstream node as a *sort point*. If all of the traffic on arc uv_1 arrives at u from s_1, then we could sort the volume at s_1 destined for v_1 into *containers* at s_1 to be unloaded at v_1 rather than to be unloaded at u. These containers are not opened at the intermediate node u, instead, they are *cross-*

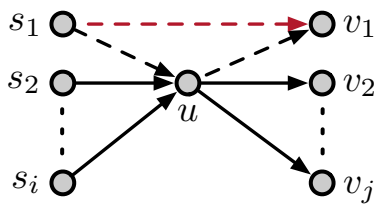


Figure 1.4

docked, an operation that is significantly faster and less costly than the process of sorting all individual parcels [58, 67].

Figure 1.4 illustrates the containerization of s_1, v_1 volume at s_1 by replacing the two dashed arcs s_1u and uv_1 by one red dashed arc s_1v_1. Note that the arc s_1u would need to remain if any packages at s_1 need to be delivered to any of $u, v_2, \ldots, v_j$. Sort points are required at each node, for each outgoing arc. Hence, the arc replacement operation in Figure 1.4 reduces the number of required sort points at node u, but increases the number of sort points required at s_1 by 1. Since sort point capacity is often limited, determining the maximum number of sort points required at any warehouse to route all commodities is important in both short- and long-term planning.

1.4.1 Formal model for sortation

Let $\mathcal{K}$ be a set of commodities, where each commodity $k \in \mathcal{K}$ has a source s_k, sink t_k, and a designated directed path P_k in D. Let the *transitive closure* of digraph D, denoted $\texttt{cl}(D)$, be the graph obtained by introducing short-cut arcs vw whenever w can be reached from v through a directed path in D. An example is shown in Figure 1.5.

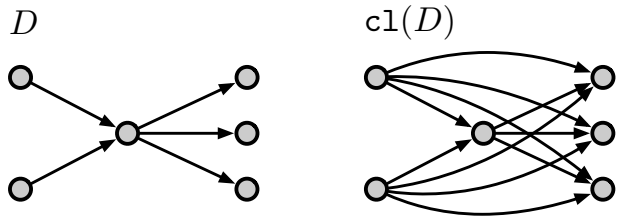


Figure 1.5: Digraph D and its transitive closure $\texttt{cl}(D)$.

We say that a subgraph H of $\texttt{cl}(D)$ is *feasible* if for each commodity $k \in \mathcal{K}$, there is an s_k, t_k-dipath in $H \cap \texttt{cl}(P_k)$.

We assume without loss of generality that D is weakly-connected (there is a path between each pair of nodes in the underlying undirected graph), the commodity set is non-empty, all commodities are non-trivial ($s_k \neq t_k$) and unique, and all nodes with no out-arcs in D serve as the sink for some commodity. Any instance can be reduced to a (set of) equivalent instances that satisfy these assumptions, by solving the instance defined on each weakly-connected component independently, and removing trivial and repeated commodities. Let $\mathcal{S} = \{s_k : k \in \mathcal{K}\}$ be the set of sources, and $\mathcal{T} = \{t_k : k \in \mathcal{K}\}$ be the set of sinks. For each node $v \in N$, let $\mathcal{T}(v) = \{t_k : (v, t_k) \in \mathcal{K}\}$ and $\mathcal{S}(v) = \{s_k : (s_k, v) \in \mathcal{K}\}$.

Two natural objectives are to (a) minimize the maximum number of sort points required at a warehouse; and to (b) minimize the total number of sort points required in the network. Objective (a) is essential to determining feasibility, or near-feasibility, of routing a given set of commodities through the network given fixed sort point capacities at the warehouses. Additionally, determining the maximum number of sort points required at a warehouse is important in planning the design of future warehouses or modification of existing warehouses. Objective (b) captures the fact that installing and running a sort point requires capital investment as well as personnel in day-to-day operations. Additionally, the number of sort points relates to dwell time and fill rate at warehouses. As the number of sort points increases in the network, we would expect the average time a package sits in a container that is waiting to be filled to increase, and the number of non-full containers shipped to increase. Thus, minimizing the total number of sort points is a natural goal in both theory and practice.

We define the min-degree sort point problem, `min-degree-SPP`, as follows.

`min-degree-SPP`: given a directed graph D and commodity set $\mathcal{K}$, find a feasible subgraph H of $\texttt{cl}(D)$ with minimum max out-degree.

Given a graph H, let $\Delta^+(H)$ denote the maximum out-degree of a node in H. Let Δ^* denote the minimum max out-degree in any feasible subgraph. In the corresponding decision version of the problem, we are given a positive integer *target*, T, and the problem is to determine whether or not $\Delta^* \leq T$.

We define the min-cardinality sort point problem, `min-cardinality-SPP`, as follows.

`min-cardinality-SPP`: given a directed graph D and commodity set $\mathcal{K}$, find a feasible subgraph H of $\texttt{cl}(D)$ with the fewest arcs.

We say that an instance $\mathcal{I} = (D, \mathcal{K})$ of `min-degree-SPP` or `min-cardinality-SPP` is a *tree instance* if the underlying undirected graph of D is a tree. More generally, for any class of graphs, $\mathcal{X}$, we say that $\mathcal{I} = (D, \mathcal{K})$ is an $\mathcal{X}$ *instance* if the underlying undirected graph of D is in $\mathcal{X}$. In this thesis, we focus on tree instances of sort point problems. For this class of instances, it suffices to ensure that a subgraph H of $\texttt{cl}(D)$ contains an s_k, t_k-dipath for each $k \in \mathcal{K}$ due to the following lemma.

Lemma 1.1. *If $\mathcal{I} = (D, \mathcal{K})$ is a tree instance, then any s_k, t_k-dipath in $\texttt{cl}(D)$ is an s_k, t_k-dipath in $\texttt{cl}(P_k)$, where P_k is the unique s_k, t_k-dipath in D.*

Proof. Since $\mathcal{I}$ is a tree instance, each arc uv in $\texttt{cl}(D)$ corresponds to a unique u, v-dipath in D. As a result, any s_k, t_k-dipath in $\texttt{cl}(D)$, Q, corresponds to an s_k, t_k-dipath in

D formed by concatenating the dipaths in D between the endpoints of the arcs in Q. Since there is a unique s_k, t_k-dipath in D, all the intermediate u, v-dipaths must be subpaths of P_k. Therefore all arcs in an s_k, t_k-dipath are in $\texttt{cl}(P_k)$ as required. □

As a result, a tree instance is uniquely determined by its graph D and set of source-sink pairs. Moreover, the visual presentation of an instance simplifies. In this thesis, when (tree) instances are given in figures, the digraph D is given and the closure is to be inferred. The nodes are numbered, and each node v is labelled with the set of indices of nodes in $\mathcal{S}(v)$. For example, the instance presented in Figure 1.6 has commodity set $\{(v_1, v_2), (v_1, v_5), (v_2, v_5), (v_3, v_6)\}$.

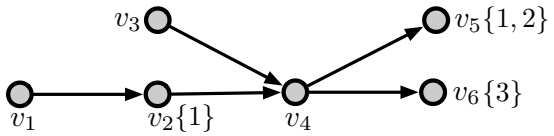


Figure 1.6: Sample tree instance.

Note that an analogous undirected version of Lemma 1.1 does not hold. Specifically, it is not the case for tree instances that all s_k, t_k-paths in the underlying undirected graph of $\texttt{cl}(D)$ are s_k, t_k-paths in the undirected graph of $\texttt{cl}(P_k)$. This observation highlights the importance of working with directed paths even in the case of tree instances.

1.5 Contributions for sortation

1.5.1 Min-degree sort point problem

First, we prove that `min-degree-SPP` is NP-hard, even when we restrict to the set of star instances.

Theorem 1.2 (See Theorem 6.1)**.** `min-degree-SPP` *is NP-hard, even when restricted to star instances.*

In order to certify the (near)-optimality of feasible solutions, we construct combinatorial lower bounds that are motivated by the witness set construction for undirected min-degree spanning trees [21]. Specifically, we define a function `LB` such that for all $W \subseteq N$ and $\mathcal{K}' \subseteq \mathcal{K}$, $\texttt{LB}(W, \mathcal{K}') \leq \Delta^*$. We show that in instances with a single source, this construction is the best possible. We develop an exact polynomial-time local search algorithm for single-source instances and certify its optimality by determining values of W and $\mathcal{K}'$ such

that $\texttt{LB}(W, \mathcal{K}') = \Delta^+(H)$ for the graph H returned. Note that the single-source setting reduces to the problem of finding a min-degree arborescence in a directed acyclic graph. This problem can be solved via bipartite b-matching in $O(n^{2/3}m \log n)$ time [22] or by a local search algorithm in $O(nm \log n)$ time [66]. Our approach uses the structure of the transitive closure, and allows us to obtain a very simple algorithm that offers a significant improvement in runtime. Moreover, it motivates our other algorithms in more complex settings that cannot be modelled as min-degree arborescence problems.

Theorem 1.3 (See Theorem 6.9)**.** *There is an $O(n \log^2 n)$-time exact algorithm for tree instances of* `min-degree-SPP` *with a single source.*

When there is a single source and the undirected graph of D is a tree, each node has at most one entering arc in D. We refer to tree instances in which each node in D has at most one entering arc as *out-tree* instances. We prove that $\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \texttt{LB}(W, \mathcal{K}') \geq \Delta^* - 1$ for out-tree instances, by showing that there is a polynomial-time algorithm that returns a solution with out-degree at most one greater than the best lower bound. We also show that there are out-tree instances where $\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \texttt{LB}(W, \mathcal{K}') = \Delta^* - 1$.

Our algorithm for multi-source out-tree instances is again combinatorial in nature, but the details are significantly more involved. In the algorithm for the single-source setting, in each iteration an arc vw is exchanged for an arc uw, where u is on the path between the root (the unique source) and v in D. However, in the multi-source setting, the same action is not sufficient to ensure a high-quality solution is found. We instead define an algorithm for `min-degree-SPP` on out-trees which takes as input a target T, and returns a feasible solution H with $\Delta^+(H) \leq T$ whenever $\Delta^* \leq T - 1$. The additional input of the target as well as a careful selection of which arcs to fix in each iteration prevent the need to backtrack. In the proof of the performance of this algorithm, we provide an explicit construction of the lower bound certificate with value at least $\Delta^* - 1$.

Theorem 1.4 (See Theorem 6.17)**.** *There is a polynomial-time additive* 1*-approximation algorithm for out-tree instances of* `min-degree-SPP`*.*

The analysis of the out-tree algorithm is *tight*, in that there are instances of `min-degree-SPP` where the algorithm does not produce an optimal solution. Moreover, the performance of the algorithm is bounded against a lower bound that has an inherent gap matching the proven performance. The algorithmic approach for the out-tree setting cannot easily be extended to more complex graph structures, such as stars, since an optimal solution is no longer guaranteed to be acyclic and our current algorithm heavily relies on this fact. Furthermore, the lower bound construction weakens significantly for star instances.

We also give a framework for obtaining approximation results for arbitrary tree instances. As a first step, we give an efficient 2-approximation when D is a star.

Theorem 1.5 (See Theorem 6.21)**.** *There is a polynomial-time 2-approximation for star instances of* `min-degree-SPP`*, and for any $\epsilon > 0$ there is a star instance of* `min-degree-SPP` *where* $\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \texttt{LB}(W, \mathcal{K}') = \frac{2}{3}\Delta^* + \epsilon$.

A generalization of the class of star instances is the class of junction trees. A tree instance is a *junction tree instance* if there is some node r in D such that r is a node in P_k for all $k \in \mathcal{K}$. Our framework for obtaining an approximation result for general tree instances builds on the approximability of junction tree instances.

Theorem 1.6 (See Theorem 6.23)**.** *Given a polytime α-approximation algorithm for junction tree instances, there is a polytime $\alpha \log n$-approximation algorithm for tree instances of* `min-degree-SPP`*.*

1.5.2 Min-cardinality sort point problem

Similar to the case of `min-degree-SPP`, when an instance has a single source, the problem of `min-cardinality-SPP` is polytime solvable. When there are multiple sources, we do not know whether or not tree instances of `min-cardinality-SPP` can be solved in polynomial time.

First, we show that an optimal solution cannot be obtained directly from solving the linear relaxation of a natural formulation. Specifically, we show that the relaxation has an integrality gap of at least $4/3 - \epsilon$ for each $\epsilon > 0$, even for path instances. Let $\mathcal{C} = \{C \subseteq N : \exists k \in \mathcal{K} \text{ such that } s_k \in C, t_k \notin C\}$. Since we focus on tree instances, any s_k, t_k-dipath in $\texttt{cl}(D)$ is a dipath in $\texttt{cl}(P_k)$. As a result, we obtain the following formulation for tree instances of `min-cardinality-SPP`.

$$\begin{array}{lll} \min & x(\texttt{cl}(D)) & \\ \text{s.t.} & x(\delta^+_{cl(D)}(C)) \geq 1, & \forall C \in \mathcal{C} \\ & x \in \{0,1\} & \end{array} \qquad \text{(IP-MCSPP)}$$

Theorem 1.7 (See Theorem 7.7)**.** *Formulation (IP-MCSPP) has an integrality gap of at least $4/3 - \epsilon$ for any $\epsilon > 0$ on path instances.*

In progress towards finding an exact algorithm for tree instances, we first present simple and efficient combinatorial algorithms for in-tree, out-tree, and star instances.

Theorem 1.8 (See Theorems 7.23 and 7.28)**.** *In-tree, out-tree, and star instances of* `min-cardinality-SPP` *can be solved in polynomial time.*

There are two key components to our combinatorial algorithms. The first component is the partitioning of an instance $\mathcal{I}$ into a collection of *weakly-connected* subproblems, $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell\}$, defined on the set of source and sink nodes that must be in the same weakly-connected component in any feasible solution. We prove that in each of the settings considered, the union of optimal solutions to each subproblem forms an optimal solution to the original instance $\mathcal{I}$. We also present an example showing that in general, this union does not necessarily form an optimal solution.

The second component is an efficient procedure that reduces the size of any tree instance until all Steiner nodes (nodes that are neither sources nor sinks) have in-degree and out-degree at least two in D. After executing this procedure on in-tree and out-tree instances, no Steiner node remains. In the case of star instances, at most one Steiner node remains after the reduction procedure. With this simplified structure we prove that optimal solutions can be determined efficiently for in-tree, out-tree, and star instances.

A byproduct of this reduction procedure is that we can bound the number of Steiner nodes remaining in a tree instance by a function of the number of source and sink nodes. As a result, we prove that the digraph obtained by running the reduction procedure has at most 3 times the number of arcs as an optimal solution.

Theorem 1.9 (See Theorem 7.13)**.** *There is a polytime 3-approximation algorithm for tree instances of* `min-cardinality-SPP`*.*

A digraph $D = (N, A)$ is a *merged-tree* if it is obtained by merging an in-tree, $T^{in} = (N^{in}, A^{in})$, and an out-tree, $T^{out} = (N^{out}, A^{out})$, at their root nodes. Let r denote the merged node in D, which we refer to as the *join*. A junction tree instance of `min-cardinality-SPP` is a merged-tree instance where the unique commodity path P_k includes the node r for each $k \in \mathcal{K}$. We prove that in order to solve a merged-tree instance $\mathcal{I} = (D, \mathcal{K})$, it suffices to find optimal solutions to a set of in-tree instances, a set of out-tree instances, and a junction tree instance. Moreover, all of these instances have the same base graph, D, and can be constructed in polynomial time.

In the following theorem, given an instance $\mathcal{I}$, $G(\mathcal{I})$ is a graph where the set of nodes in the same connected component are sources and sinks in $\mathcal{I}$ that must be in the same weakly-connected component in any feasible solution. Given a partitioning $\mathcal{N}$ of N, the function $\phi_{\mathcal{N}}(\mathcal{I})$ maps a given junction tree instance to a junction tree instance on the same graph where all sources and sinks are moved to the node in the same part in $\mathcal{N}$ that is

nearest to r in D. These functions are defined formally in Chapter 7.

Theorem 1.10 (See Theorem 7.34)**.** *Let $\mathcal{I} = (D, \mathcal{K})$ be a merged-tree instance of* `min-cardinality-SPP` *with join r. Let $\mathcal{I}_{\mathcal{L}} = (D, \mathcal{K}_{\mathcal{L}})$ and $\mathcal{I}_{\mathcal{J}} = (D, \mathcal{K}_{\mathcal{J}})$ where*

$$\mathcal{K}_{\mathcal{L}} = \{k \in \mathcal{K} : r \notin P_k\} \text{ and } \mathcal{K}_{\mathcal{J}} = \{k \in \mathcal{K} : r \in P_k\}.$$

Let $\mathcal{N} = \{N_1, N_2, \ldots, N_\ell\}$ be the node sets of the maximally-connected components in $G(\mathcal{I}_{\mathcal{L}})$. If H_1 and H_2 are optimal solutions for $\mathcal{I}_{\mathcal{L}}$, and $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$ respectively, and H_1 is minimally-crossing, then $H_1 \cup H_2$ is an optimal solution for $\mathcal{I}$.

As in the case of `min-degree-SPP` it remains open to find a "good" algorithm (possibly an exact algorithm in this case) for junction tree instances. We present an efficient algorithm for the special case of spider instances.

Theorem 1.11 (See Theorem 7.46)**.** *Spider instances of* `min-cardinality-SPP` *can be solved in polynomial time.*

The proof of correctness relies on working with *minimally-crossing* solutions. We say that two arcs v_1w_1 and v_2w_2 in $\texttt{cl}(D)$ are *crossing* if the unique v_1, w_1- and v_2, w_2-dipaths in D share an arc. We present uncrossing operations that allow us to move from one optimal solution to another, potentially with fewer crossing arcs. As a byproduct of the additional structure of minimally-crossing solutions, we prove that it is equivalent to solve an instance with additional commodities. By adding a particular set of commodities to spider instances, we show that the resulting spider instance decomposes into instances $\mathcal{I}_{\mathcal{L}}$ and $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$ that can be solved by the in-tree, out-tree, and star algorithms.

1.5.3 Related work

Degree-bounded network design problems are fundamental and well-studied combinatorial optimization problems. A prominent example is the *minimum-degree spanning tree problem* which asks, given an undirected, unweighted graph, for a spanning tree with minimum maximum degree. Fürer and Raghavachari [21] introduced the problem and presented a local-search based polynomial-time algorithm for computing a spanning tree with maximum degree which is at most 1 larger than the optimal maximum degree. Their combinatorial arguments rely on *witness sets* chosen from a family of carefully-constructed lower bounds [9, 65].

Various techniques have been employed in subsequent work on the weighted setting, e.g., [24, 34, 36, 51, 8, 7, 52], culminating in the result that one can compute a spanning tree of

minimum cost that exceeds the degree bound by at most 1 [56]. Since then, generalizations have been studied such as the *degree-bounded Steiner tree* problem [35, 42], *survivable network design* with higher connectivity requirements [41, 42] and the *group Steiner tree* problem [14, 25, 37].

Directed (degree-bounded) network design problems are typically substantially harder than their undirected counterparts. Among the few nontrivial approximation results are quasipolynomial-time bicriteria approximations (with respect to cost and maximum out- or in-degree) [25] for the degree-bounded directed Steiner tree problem and approximation results for problems with intersecting or crossing supermodular connectivity requirements [1, 47].

A special case in directed degree-bounded network design is the *min-degree arborescence problem* where, given a directed graph and root r, the goal is to find a spanning tree rooted at r with minimum max out-degree. This problem is NP-hard [1, 21, 41] in general, and polytime solvable in directed acyclic graphs [66].

1.6 Organization

Part I focuses on solving temporal network design problems via the solution technique of dynamic discretization discovery (DDD). In Chapter 3 we present a DDD framework where the set of departure times is determined on the arc level rather than the node level. We apply this arc-based DDD method to instances of the service network design problem (SND). We show that an arc-based approach is particularly advantageous when instances arise from region-based networks, and when candidate paths are fixed in the base graph for each commodity.

In Chapter 4, we present bounds on the additional storage that must be permitted in each iteration. Our arguments rely solely on the structure of of the standard map, μ, from the original formulation to the smaller relaxed formulations. Since our analysis only relies on μ, it generalizes to fixed-charge problems, including the service network design problem. We demonstrate our techniques in the case of the classical universal packet routing (UPR) problem in the presence of bounded node storage. Focusing on UPR allows us to clearly present the extension of the DDD method to a problem with bounded node storage without complicating constraints already addressed by previous work. We also present computational results demonstrating the effectiveness of DDD when solving universal packet routing.

In Chapter 5 we present a DDD model for solving SND with cyclic constraints. In this

process, we define a new condition on the set of timed nodes required in a partial network in each iteration of DDD. We also show that certain policies for removing timed nodes from the current partial network can result in iteration cycling.

Part II focuses on the selection of sort points required to route a given set of commodities with designated routes.

In Chapter 6 we provide exact and approximation polynomial time algorithms for the min-degree sort point problem. When the underlying network is acyclic, we propose algorithms that are fast and build on combinatorial *witness set* type lower bounds that are reminiscent and extend those used in earlier work on degree-bounded spanning trees and arborescences.

In Chapter 7 we present exact and approximation polynomial time algorithms for the min-cardinality sort point problem. We introduce reduction, decomposition, and uncrossing techniques that simplify input instances. Using these techniques, we present fast combinatorial algorithms for solving in-tree, out-tree, and spider instances. We show that the natural LP relaxation has an integrality gap of at least $\frac{4}{3} - \epsilon$ for any $\epsilon > 0$, as well as a 3-approximation algorithm for tree instances.

In Chapter 8 we present a summary of the contributions in this thesis as well as a set of open problems.

Part I

DDD

Chapter 2

Preliminaries

The first focus of this thesis is the advancement of the DDD framework, a paradigm which applies to a broad class of temporal network design problems. The DDD framework was first applied to the *service network design (SND)* problem, a problem that is frequently used to model the coordination of the physical route of the shipments as well as the departure time for each leg of the journey. In practical applications, there are often additional restrictions on the physical routes of each shipment. We will refer to SND with the addition that each commodity has a designated feasible subgraph in space as *service network design with restricted routes* (SND-RR). We will introduce the DDD paradigm by presenting a generic DDD algorithm for SND-RR.

Time-expanded networks

A time-indexed formulation for a (discrete-time) temporal problem is obtained by expressing the temporal problem as a static problem in the corresponding *time-expanded network*.

Let $D = (N, A)$ be a directed graph with nodes N and arcs A, along with arc transit times τ. Let T be the time horizon under consideration. The corresponding *(fully) time-expanded network*, $D_T = (N_T, A_T \cup H_T)$, consists of a copy of each node $v \in N$ for each time point $t \in [T] := \{0, 1, \ldots, T\}$. The *movement* arcs, A_T, consist of a copy of each arc in A for each departure time, and the *holdover* arcs, H_T, connect each node copy to the

next consecutive copy. Specifically,

$$N_T := \{(v,t) : v \in N, t \in [T]\},$$
$$A_T := \{((v,t),(w,t+\tau_{vw})) : (v,t) \in N_T, vw \in A, t+\tau_{vw} \leq T\}, \text{ and}$$
$$H_T := \{((v,t),(v,t+1)) : v \in N, t \in [T-1]\}.$$

An example is presented in Figure 2.1. We will often refer to arcs and nodes in a time-expanded network as *timed arcs* and *timed nodes* respectively. We will write timed nodes with their associated times as (v,t), and the corresponding node in the base graph is v. Similarly, we write timed arcs as $((v,t),(w,t'))$, and the corresponding arc in the base graph as vw. Additionally, we will refer to paths in the time-expanded network as *trajectories*. We will often refer to paths in D as *flat paths*.

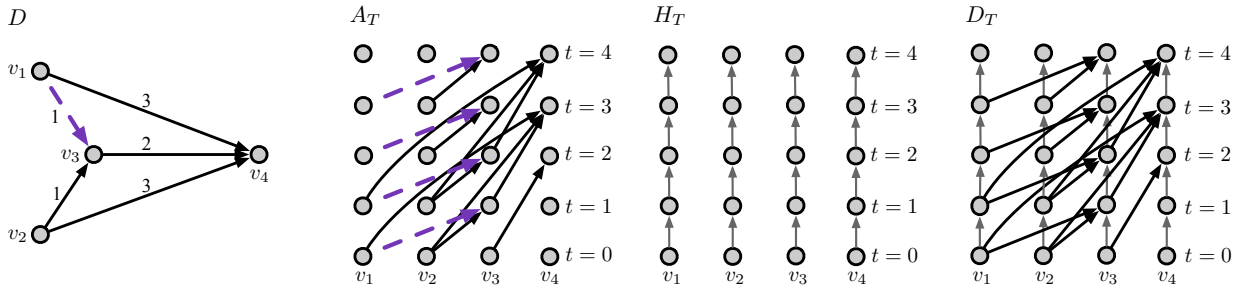


Figure 2.1: Base graph D and the construction of D_T when $T = 4$.

A temporal network design problem with time horizon T can be modelled as a static network design problem on D_T after assigning appropriate costs, capacities, and demands. However, while time-indexed formulations have strong linear relaxations, these formulations become impractical to solve for fine time discretizations since the number of variables, reflected in the size of D_T, grows linearly in T [19, 20].

Service Network Design

In an instance of *service network design* (SND), we are given a directed graph $D = (N, A)$ with node set N and arc set A, and a set of commodities $\mathcal{K}$. We refer to D as the *flat network*. Each arc $vw \in A$ has an associated transit time $\tau_{vw} \in \mathbb{N}_{>0}$, a commodity-dependent per-unit-of-flow cost $c^k_{vw} \in \mathbb{R}_{>0}$ for each $k \in \mathcal{K}$, a fixed cost $f_{vw} \in \mathbb{R}_{>0}$, and a capacity $u_{vw} \in \mathbb{N}_{>0}$. We interpret u_{vw} as the capacity of trucks scheduled on arc vw, and f_{vw} as the per-truck-cost on vw. Each commodity $k \in \mathcal{K}$ has a source $o_k \in N$ and sink $d_k \in N$, along with a demand q_k that must be routed along a single trajectory from o_k to d_k

(i.e. it is unsplittable). Let r_k denote the time commodity k becomes available at its origin (its release time) and let l_k denote the deadline for commodity k. Let $T = \max_{k \in \mathcal{K}} l_k$ be the latest deadline among the commodities. We may assume that the earliest release time is time 0 by shifting the time horizon. An o_k, d_k-trajectory in D_T is *feasible* for commodity k if it departs o_k no earlier than r_k and arrives at d_k no later than l_k. The goal of SND is to determine a feasible trajectory in D_T for each commodity in order to minimize the total fixed and variable cost of the resources required to ship the commodities along the trajectories.

We define *service network design with restricted routes* (SND-RR) as the problem of SND with the additional input of a designated subgraph $D^k \subseteq D$ for each commodity $k \in \mathcal{K}$. In SND-RR, an o_k, d_k-trajectory Q in D_T is *feasible* for commodity k if it departs o_k no earlier than r_k, arrives at d_k no later than l_k, and the flat path corresponding to the trajectory Q is fully contained in D^k.

Time-indexed formulation for SND-RR

We now define a time-indexed formulation for SND-RR analogous to the formulation for SND presented by Boland et al. in [2]. For each $k \in \mathcal{K}$, let $D_T^k = (N_T^k, A_T^k \cup H_T^k)$ denote the time-expanded network for D^k with time horizon T. For each commodity $k \in \mathcal{K}$ and each timed arc $a \in A_T^k \cup H_T^k$, let x_a^k be a binary variable which is equal to 1 if commodity k is scheduled to travel along timed arc a in its assigned trajectory. For each $a \in A_T$, let y_a denote the number of trucks scheduled along timed arc a. For each timed arc $a = ((v,t),(w,t')) \in A_T$, let $u_a := u_{vw}$, $c_a := c_{uv}$, and $f_a := f_{uv}$.

Let $\delta^+_{D_T}(v,t)$ and $\delta^-_{D_T}(v,t)$ denote the outgoing and incoming timed arcs at (v,t) in D_T. That is, $\delta^+_{D_T}(v,t) = \{a \in A_T \cup H_T : a = ((v,t),(w,t'))\}$, and $\delta^-_{D_T}(v,t) = \{a \in A_T \cup H_T : a = ((w,t'),(v,t))\}$.

The following integer program (IP) models SND-RR.

$$\min \quad \sum_{a \in A_T} f_a y_a + \sum_{a \in A_T} \sum_{k \in \mathcal{K}} c_a^k q_k x_a^k \qquad \text{(SND-RR}(D_T)\text{)}$$

$$\text{s.t.} \quad x^k(\delta^+_{D_T^k}(v,t)) - x^k(\delta^-_{D_T^k}(v,t)) = \begin{cases} 1 & (v,t) = (o_k, r_k) \\ -1 & (v,t) = (d_k, l_k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_T^k \quad (2.1)$$

$$\sum_{k \in \mathcal{K}} q_k x_a^k \leq u_a y_a \quad \forall a \in A_T \qquad (2.2)$$

$$x_a^k \in \{0,1\} \quad \forall k \in \mathcal{K}, a \in A_T^k \cup H_T^k \qquad (2.3)$$

$$y_a \in \mathbb{N}_{\geq 0} \quad \forall a \in A_T \qquad (2.4)$$

Constraint (2.1) ensures that each commodity is assigned a feasible trajectory. Constraint (2.2) ensures that for each timed arc there is sufficient capacity purchased to accommodate the trajectories scheduled to traverse that timed arc. The objective function optimizes for the total fixed and variable cost. Note that when $D^k = D$ for all $k \in \mathcal{K}$, this is the same as the timed-indexed formulation for SND in [2].

2.1 Generic DDD framework

The downside to modelling temporal problems with time-expanded networks is that this time-expansion often makes the resulting static problem impractical to solve since the network grows linearly in T [19, 20]. In an effort to avoid solving these large time-indexed formulations, a DDD algorithm solves a series of mixed-integer programs (MIPs) defined on smaller *partially time-expanded networks* that appropriately *underestimate* the transit times of the arcs in A_T.

Definition 2.1. *A* partially time-expanded network *with respect to D and T is any directed graph $D_S = (N_S, A_S \cup H_S)$ where $N_S \subseteq N_T$,*

$$A_S \subseteq \{((v,t),(w,t')) : (v,t),(w,t') \in N_S, vw \in A, t' \leq t + \tau_{vw}\},$$

and H_S connects each node copy to its next copy in N_S.

We will refer to fully time-expanded networks and partially time-expanded networks as *full* and *partial* networks respectively. In this paper, $D_T = (N_T, A_T \cup H_T)$ denotes the full network with time horizon T, and $D_S = (N_S, A_S \cup H_S)$ is any partial network.

In general, a DDD algorithm aims to solve a instance $\mathcal{I}$ of a minimization problem P defined on D_T, with a corresponding mixed integer program (MIP), IP(D_T). Partial networks are constructed so that an optimal solution to the MIP induced by D_S, denoted IP(D_S), provides a lower bound on the optimal value of IP(D_T). Moreover the partial networks are constructed and refined in such a way that they are sparse relative to the full network.

Algorithm 1: `Generic-DDD`($\mathcal{I}, D_T$)

Input: Instance $\mathcal{I}$ of problem P defined on D_T

Initialization: $D_S \leftarrow D_0$, where IP(D_S) provides a lower bound for IP(D_T)

while *not solved* **do**

  Lower bound: Solve IP(D_S) and obtain a solution $\hat{x}$ in D_S

  Upper bound/termination: determine if $\hat{x}$ can be converted to a solution to IP(D_T) with equal cost

  if *yes* **then**

    Stop. An optimal solution has been found for IP(D_T).

  Refinement: update D_S while ensuring IP(D_S) provides a lower bound for IP(D_T).

2.2 Generic DDD for SND-RR

In the context of SND-RR, we denote the lower bound formulation corresponding to a partial network D_S as SND-RR(D_S). We define SND-RR(D_S) so that it is the formulation corresponding to routing the commodities in D_S rather than D_T, with the same induced costs.

Let $\mathcal{I}$ be an instance of SND-RR with base graph $D = (N, A)$, commodity set $\mathcal{K}$, and designated subgraph $D^k = (N^k, A^k) \subseteq D$ for each $k \in \mathcal{K}$. Let $D_S = (N_S, A_S \cup H_S)$ be a partial network. For each $k \in \mathcal{K}$ let D_S^k be the subgraph of D_S with node set

$$N_S^k := \{(v,t) \in N_S : v \in N^k\},$$

and arc set $A_S^k \cup H_S^k$ where

$$A_S^k := \{((v,t),(w,t')) \in A_S : vw \in A^k\} \text{ and } H_S^k := \{((v,t),(v,t')) \in H_S : v \in N^k\}.$$

For each timed arc $a = ((v,t),(w,t')) \in A_T$, let $\tau_a := \tau_{vw}$. The following formulation is analogous to the lower bound formulation presented in Boland et al. [2].

$$\min \quad \sum_{a \in A_S} f_a y_a + \sum_{a \in A_S} \sum_{k \in \mathcal{K}} c_a^k q_k x_a^k \qquad \text{(SND-RR}(D_S)\text{)}$$

$$\text{s.t.} \quad x^k(\delta^+_{D_S^k}(v,t)) - x^k(\delta^-_{D_S^k}(v,t)) = \begin{cases} 1 & (v,t) = (o_k, r_k) \\ -1 & (v,t) = (d_k, l_k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_S^k \quad (2.5)$$

$$\sum_{k \in \mathcal{K}} q_k x_a^k \leq u_a y_a \quad \forall a \in A_S \qquad (2.6)$$

$$\sum_{a \in A_S^k} \tau_a x_a^k \leq l_k - r_k, \quad \forall k \in \mathcal{K}. \qquad (2.7)$$

$$x_a^k \in \{0,1\} \quad \forall k \in \mathcal{K}, a \in A_S^k \cup H_S^k \qquad (2.8)$$

$$y_a \in \mathbb{N}_{\geq 0} \quad \forall a \in A_S \qquad (2.9)$$

Constraint (2.5) ensures that each commodity is assigned a feasible trajectory in D_S. Constraint (2.6) ensures that for each timed arc there is sufficient capacity purchased to accommodate the trajectories scheduled to traverse that timed arc. The addition of constraint (2.7) enforces that each commodity is assigned to traverse a flat path in D with a feasible transit time (at most $l_k - r_k$). The objective function is analogous to that of (SND-RR(D_T)).

If a solution to the partial network cannot be converted to a solution to the original instance of equal cost, the partial network is refined by adding timed nodes and timed arcs based on the current solution. An overview of this method is presented in Algorithm 2.

Algorithm 2: `Generic-DDD`$(D, \mathcal{K})$

Input: Base network $D = (N, A)$, commodity set $\mathcal{K}$ with subgraph D^k for each $k \in \mathcal{K}$

Initialization: $D_S \leftarrow D_0$, where SND-RR(D_S) provides a lower bound for SND-RR(D_T)

while *not solved* **do**

  Lower bound: Solve SND-RR(D_S) and obtain a solution $(\hat{x}, \hat{y})$ in D_S

  Upper bound/termination: determine if $\hat{x}$ can be converted to a solution to SND-RR(D_T) with equal cost

  if *yes* **then**

    Stop. An optimal solution has been found for SND-RR(D_T).

  Refinement: update D_S while ensuring SND-RR(D_S) provides a lower bound for SND-RR(D_T).

We now describe the specific initialization, lower and upper bound, and refinement steps in the original DDD implementation for SND presented in [2]. The only change is that instead of solving SND(D_S) in each iteration, we solve SND-RR(D_S). Note, SND(D_S) is equivalent to SND-RR(D_S) when $D^k = D$ for all $k \in \mathcal{K}$. We will refer to this DDD algorithm as the *node-based* DDD approach.

Initialization and lower bound

When $D^k = D$ for all $k \in \mathcal{K}$, Boland et al. prove that so long as the partial network D_S satisfies the following two properties, an optimal solution of SND-RR(D_S) provides a lower bound on the optimal value of SND-RR(D_T) (Theorem 2 in [2]).

(P1) **Timed nodes**: For all $k \in \mathcal{K}$, (o_k, r_k) and (d_k, l_k) are in N_S;
For all $v \in N$, $(v, 0)$, and (v, T) are in N_S;

(P2) **Arc copies**: For all $vw \in A$, for all $(v, t) \in N_S$ with $t + \tau_{vw} \leq T$, we have $((v, t), (w, t')) \in A_S$ where $t' = \max\{r : r \leq t + \tau_{vw}, (w, r) \in N_S\}$.

Note that property (P2) gives a natural method to construct D_S given N_S. In its standard implementation, in the first iteration of a DDD algorithm the partial network is initialized to be the network consisting only of the timed nodes stated in (P1). Throughout this thesis we will assume that any partial network has no additional arcs beyond those stated in (P2). Thus, the partial networks we work with are completely determined by the flat network and the set of timed nodes.

The following generalization of Theorem 2 in [2] follows directly from a restatement of the proof and the observation that physical routes of each commodity are unchanged.

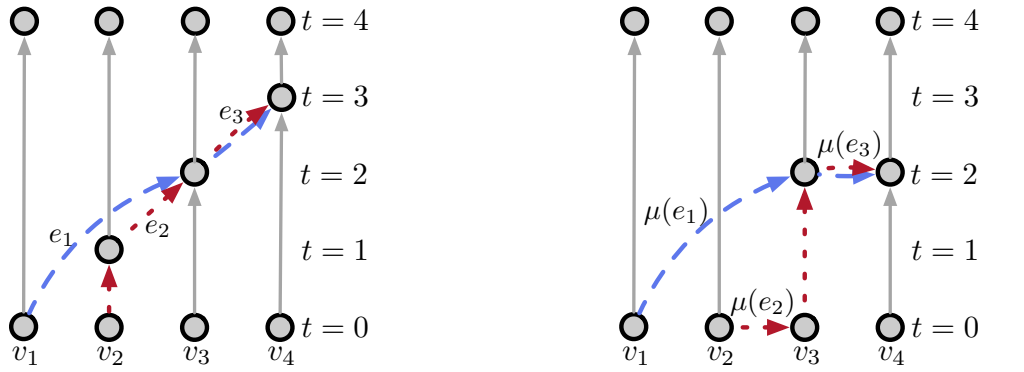
Theorem 2.2. *Let $\mathcal{I}$ be an instance of SND-RR with base graph $D = (N, A)$, commodity set $\mathcal{K}$, and designated subgraph $D^k \subseteq D$ for each $k \in \mathcal{K}$. Let $D_S = (N_S, A_S \cup H_S)$ be a partial network that satisfies properties $(P1)$ and $(P2)$. Then an optimal solution to SND-RR(D_S) provides a lower bound on the optimal value of SND-RR(D_T).*

Proof. Let $\mu : A_T \to A_S$ be the map defined so that for each timed arc $a \in A_T$,

$$a = ((v, t), (w, t')) \quad \to \quad \mu(a) = ((v, \hat{t}), (w, \hat{t}')), \tag{2.10}$$

where $\hat{t} = \max\{s : s \leq t, (v, s) \in N_S\}$, and $\hat{t}' = \max\{s : s \leq \hat{t} + \tau_{vw}, (v, s) \in N_S\}$. Note that μ is well-defined since $(P1)$ ensures that there is a timed copy of each node in N at time 0, and $(P2)$ ensures that there is a (unique) copy of each arc in A departing the selected timed node. Moreover, this timed arc underestimates the correct transit time.

Examples of the map μ are given in Figure 2.2, where Figure 2.2a shows the original timed arcs on a subgraph of D_T, and Figure 2.2b shows the resulting timed arcs after applying μ, given a partial network with the timed node set as shown.



(a) Two trajectories in D_T.

(b) Corresponding trajectories in D_S.

Figure 2.2

Let $(\bar{x}, \bar{y})$ be a feasible solution to (SND-RR(D_T)) with cost C, and let $\mathcal{Q} = \{Q_k\}_{k \in \mathcal{K}}$ denote the corresponding set of trajectories. Fix $k \in \mathcal{K}$. The set of ordered movement arcs in Q_k is $\{a_1, a_2, \ldots, a_{q_k}\}$, where each $a_i = ((v_i, t_i^{out}), (v_{i+1}, t_{i+1}^{in}))$ for each $i \in [q_k - 1]$. Since Q_k is feasible, the following statements are true:

(S1) $v_1 = o_k$, and $t_1^{out} \geq r_k$
(S2) $v_{q_k} = d_k$, and $t_{q_k}^{in} \leq l_k$
(S3) $t_{i+1}^{in} = t_i^{out} + \tau_{v_i v_{i+1}}$ for all $i \in [q_k - 1]$
(S4) $t_i^{in} \leq t_i^{out}$ for all $i \in \{2, 3, \ldots, q_k - 1\}$

Consider the ordered set of timed arcs $\{\mu(a_1), \mu(a_2), \ldots, \mu(a_{q_k})\}$ in A_S, where $\mu(a_i) = ((v_i, \hat{t}_i^{out}), (v_{i+1}, \hat{t}_{i+1}^{in}))$ for each $i \in [q_k - 1]$.

Claim 1: $\hat{t}_i^{in} \leq \hat{t}_i^{out}$ for all $i \in \{2, 3, \ldots, q_k - 1\}$.

Fix $i \in \{2, 3, \ldots, q_k - 1\}$. By definition, $\hat{t}_i^{in} = \max\{t : t \leq \hat{t}_{i-1}^{out} + \tau_{v_{i-1}v_i}, (v_i, t) \in N_S\}$ where $\hat{t}_{i-1}^{out} = \max\{t : t \leq t_{i-1}^{out}, (v_{i-1}, t) \in N_S\}$. Thus,

$$\hat{t}_i^{in} \leq \max\{t : t \leq t_{i-1}^{out} + \tau_{v_{i-1}v_i}, (v_i, t) \in N_S\}$$

Similarly, $\hat{t}_i^{out} = \max\{t : t \leq t_i^{out}, (v_i, t) \in N_S\}$. Since $t_i^{in} = t_{i-1}^{out} + \tau_{v_{i-1}v_i}$, and $t_i^{in} \leq t_i^{out}$, it follows that

$$\hat{t}_i^{out} \geq \max\{t : t \leq t_{i-1}^{out} + \tau_{v_{i-1}v_i}, (v_i, t) \in N_S\},$$

which gives the desired result that $\hat{t}_i^{in} \leq \hat{t}_i^{out}$.

Thus, we can obtain trajectories $\hat{\mathcal{Q}} = \{\hat{Q}_k\}_{k \in \mathcal{K}}$ in D_S by mapping each movement arc $a \in A_T$ to $\mu(a) \in A_S$, and forming trajectories by adding holdover arcs. Figure 2.2 shows two examples of this map from trajectories in D_T to D_S.

Furthermore, the map μ preserves the underlying arc, and so since Q_k is a trajectory in D_T^k, it follows that $\hat{Q}_k$ is a trajectory in D_S^k. By (S1) and (S2), it follows that $\hat{t}_1^{out} \geq r_k$ and $\hat{t}_{q_k} \leq l_k$. Therefore $\hat{Q}_k$ is feasible in D_S for commodity k.

Finally, the underlying paths in the flat network for $\hat{\mathcal{Q}}$ and $\mathcal{Q}$ are the same for each commodity, and any pair of commodities k, j traversing the same timed arc $a \in A_T$ now traverse the same timed arc $\mu(a) \in A_S$. Therefore the cost incurred when routing $\hat{\mathcal{Q}}$ in D_S by each arc in A is at most the cost incurred by that arc when routing $\mathcal{Q}$ in D_T, and so the cost of $\hat{\mathcal{Q}}$ is at most C. □

Upper bound and refinement

Let $(\hat{x}, \hat{y})$ be an optimal solution to SND-RR(D_S) with corresponding trajectories $\mathcal{Q} = \{Q_k\}_{k \in \mathcal{K}}$ in D_S, and paths $\mathcal{P} = \{P_k\}_{k \in \mathcal{K}}$ in D. We obtain a feasible solution to SND-RR(D_T) with the same flat paths $\mathcal{P}$ (and hence same variable cost as $\mathcal{Q}$) by specifying a departure time for each arc in P_k, for each commodity $k \in \mathcal{K}$, that satisfies the release time

and deadline of k and respects the actual transit time of the arcs. The feasible solution also has the same fixed cost as $\mathcal{Q}$ if every pair of commodities dispatched together on an arc vw in $\mathcal{Q}$ still shares a common departure time for arc vw. In this case, the truck assignment on each arc vw for $\mathcal{Q}$ can be shifted in time to route all flow on vw in the new feasible solution. Thus, the paths $\mathcal{P}$ along with a set of departure times that satisfies the aforementioned feasibility and consolidation requirements provides an optimal solution to SND-RR(D_T).

Due to the inclusion of the path transit time constraint, Constraint (2.7), there is a set of departure times that is feasible for the set of paths $\mathcal{P}$ proposed in each iteration (satisfies release times, deadlines, and actual transit times). That is, for each trajectory Q_k there is a feasible trajectory in D_T with the same underlying path in the flat network.

The continuous formulation presented in [2] has a variable t_v^k for each commodity $k \in \mathcal{K}$ and each non-destination node v in P_k that models the departure time for k on the arc leaving v in P_k. The feasibility of the departure times is captured in the following constraints, where for fixed $k \in \mathcal{K}$, v_p is the pth node in P_k.

$$t^k_{v_p} + \tau_{v_p v_{p+1}} \leq t^k_{v_{p+1}} \quad \forall k \in \mathcal{K}, p \in [|P_k| - 1] \tag{2.11}$$

$$r_k \leq t^k_{o_k} \quad \forall k \in \mathcal{K} \tag{2.12}$$

$$t_{v_{|P_k|-1}} + \tau_{v_{|P_k|-1} d_k} \leq l_k \quad \forall k \in \mathcal{K}, \tag{2.13}$$

For each arc $vw \in A$, let J_{vw} denote the set of pairs of commodities that traverse arc vw at the same time in $\mathcal{Q}$ (i.e. use the same timed copy of vw in $\mathcal{Q}$), and let $\mathcal{J} := \{J_{vw}\}_{vw \in A}$. Boland et al. introduce a variable $\delta^{k_1 k_2}_{vw}$ for each pair of commodities $(k_1, k_2) \in J_{vw}$, for each arc $vw \in A$, along with the following set of constraints.

$$\delta^{k_1 k_2}_{vw} \geq t^{k_1}_v - t^{k_2}_v \quad \forall (k_1, k_2) \in J_{vw}, \forall vw \in A \tag{2.14}$$

$$\delta^{k_1 k_2}_{vw} \geq t^{k_2}_v - t^{k_1}_v \quad \forall (k_1, k_2) \in J_{vw}, \forall vw \in A. \tag{2.15}$$

Observe that $\delta^{k_1 k_2}_{vw}$ must always be non-negative, and if $\delta^{k_1 k_2}_{vw}$ is zero then the departure times for commodities k_1 and k_2 on arc vw are the same. Thus, if there is a vector $(\bar{t}, \bar{\delta})$ that satisfies constraints (2.11)-(2.15) with $\bar{\delta} = \mathbf{0}$, then we have found an optimal solution to SND-RR(D_T). If not, then the current partial network, D_S, must be modified.

To update the partial network, we want to add a small number of timed nodes that correct the current set of infeasible trajectories. Boland et al. stipulate that in the continuous formulation, the departure times for commodities with feasible trajectories in $\mathcal{Q}$ (those that consisted only of timed arcs with transit times equal to the transit times of the

corresponding flat arcs), are unchanged. Let $\mathcal{K}^F \subseteq \mathcal{K}$ denote the set of commodities with feasible trajectories in $\mathcal{Q}$, and let $\mathcal{T}^F = \{\mathcal{T}_k\}_{k\in\mathcal{K}^F}$ where $\mathcal{T}_k = \{\bar{t}^k_{v_p}\}_{p\in[|P_k|-1]}$ is the set of departure times for the arcs in P_k given by $\mathcal{Q}$. The following constraints ensure the trajectories for the set $\mathcal{K}^F$ are unchanged.

$$t^k_{v_p} = \bar{t}^k_{v_p}, \quad \forall k \in \mathcal{K}^F, p \in [|P_k| - 1] \tag{2.16}$$

In total, the continuous formulation presented in [2] is as follows.

$$\begin{aligned} \min \quad & \sum_{vw\in A} \sum_{(k_1,k_2)\in J_{vw}} \delta^{k_1k_2}_{vw} && \text{(LP-UB}(\mathcal{P},\mathcal{J},\mathcal{K}^F,\mathcal{T}^F)) \\ \text{s.t.} \quad & (2.11) - (2.16) \end{aligned}$$

Due to the addition of constraint (2.7), in each iteration there is a feasible solution $(\bar{t}, \bar{\delta})$ to the continuous formulation LP-UB$(\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F)$, which corresponds to a feasible solution $(\bar{x}, \bar{y})$ to SND-RR(D_T). If $\bar{\delta} = \mathbf{0}$, then $(\bar{x}, \bar{y})$ is an optimal solution to SND-RR(D_T). Otherwise, a non-empty subset of the δ-variables have positive value. For each such variable $\delta^{k_1k_2}_{vw}$ with positive value, one of k_1 and k_2 must be a trajectory in D_S which has a short timed arc by construction of $\mathcal{K}^F$ and the addition of constraint (2.16). Let

$$\mathcal{C} := \{k_1 \in \mathcal{K} \setminus \mathcal{K}^F : \exists vw \in A, k_2 \in \mathcal{K}, \bar{\delta}^{k_1k_2}_{vw} > 0\}$$

capture this set of commodities. We see that $\mathcal{C}$ is a non-empty set, and each commodity in $\mathcal{C}$ has a trajectory in D_S defined by $\hat{x}$ with a short timed arc. In the refinement step, for each $k \in \mathcal{C}$ a timed node is added to lengthen the short timed arc with the earliest departure time in the trajectory Q_k.

2.3 Limitations of the construction of D_S

The goal in designing a DDD algorithm is that an optimal solution can be found in less time that it would take to solve the full time-indexed formulation. A DDD algorithm is likely to perform well if there are not too many iterations until termination, and the corresponding formulations are (very) small compared to the size of the full time-indexed formulation. A reasonable proxy for the size of the lower bound formulation solved in each iteration is the size of the partial network, D_S.

In the existing DDD framework, in each iteration, all arcs departing a common node share the same set of permissible departure times. This is problematic for temporal problems where all near-optimal solutions require a large number of departure times at many nodes

in the network. For such problems, in the final iteration of DDD, D_S must contain a large proportion of the timed nodes in D_T. Thus, in the current implementation of DDD this leads to a final formulation with a size close to that of the time-indexed formulation on the fully time-expanded network. Region-based networks are one such structure that often leads to many high-degree nodes, and their increasing popularity underscores the importance of tailoring solution methods for these networks.

A second challenge is handling static storage constraints without leading to a weak relaxation in each iteration. Since packages destined for the same outbound truck leaving a node u may arrive at different times, consolidation often necessitates the storage of packages at warehouses. Thus, bounding the additional storage required at a timed node in a partial network is essential to defining a valid lower bound.

Chapter 3

Arc-based DDD

In the current implementation of DDD, in each iteration all arcs departing a common node share the same set of permissible departure times. This causes DDD to be ineffective for solving problems where all near-optimal solutions require many distinct departure times at the majority of the high-degree nodes in the network. We develop a DDD framework where the set of departure times is determined on the arc level rather than the node level, and apply our approach to instances of SND. Our algorithm builds upon the existing DDD framework and achieves these improvements with only simple modifications to the original implementation. We show that an arc-based approach is particularly advantageous when instances are defined on hub-and-spoke networks, and when candidate paths are fixed in the base graph for each commodity.

3.1 Auxiliary graph and formulation for SND-RR

We first present examples demonstrating how the node-based DDD approach can lead to dense partial networks with large corresponding formulations. We then highlight the challenges in creating sparser partial networks using the existing construction of partial networks. Finally, we present a new auxiliary network and model for SND-RR that serves as the base for our sparser DDD approach in Section 3.2.

Consider a problem for which all optimal solutions require many departure times at a large number of nodes. For such a problem, in order to express an optimal solution on a partial network $D_S = (N_S, A_S \cup H_S)$, N_S must include a large portion of N_T. In each iteration, the partial network D_S is constructed given the set N_S according to property (P2):

(P2) For all $vw \in A$, for all $(v,t) \in N_S$ with $t + \tau_{vw} \leq T$, we have $((v,t),(w,t')) \in A_S$ where

$$t' = \max\{r : r \leq t + \tau_{vw}, (w,r) \in N_S\}.$$

Since this construction results in a timed copy of each arc in $vw \in A$ for each timed copy of v in N_S, when N_S is large, this forces A_S to be large as well. As a result, solving the lower bound formulation in the final iteration would be comparable to solving the full time-indexed formulation in the first place.

For example, consider the graph structure in Figure 3.1 with m arcs departing a node v, all with unit transit times. For each $i \in [m]$, suppose there is a commodity k with origin v, destination v_i, and release time i. Due to the current structure of partial networks, the initial partial network $D_0 = (N_0, A_0 \cup H_0)$ would have a copy of node v at all times in $[m]$. As a result, D_0 would have a copy of each arc for each departure time in $[m]$. Thus, we would have $A_0 \approx A_T$, and so the size of SND-RR(D_S) is approximately the same as SND-RR(D_T). However, observe that an optimal solution can be expressed on a small subgraph of D_T: include a single copy of arc vv_i departing v at time i for each $i \in [m]$. This simple instance highlights the issue with high-degree nodes in the network and the current construction of partial networks. Extensions of the original node-based DDD algorithm in [30, 45] consist of refinement processes that can lead to similar problematic structures.

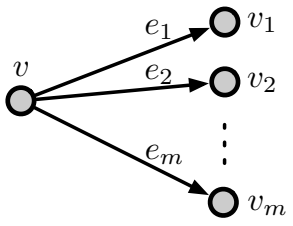
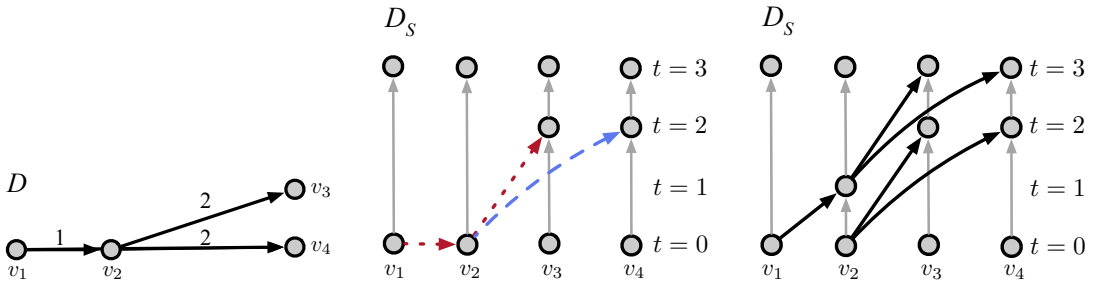


Figure 3.1

The refinement procedure in the generic node-based DDD algorithm also contributes to dense partial networks. Consider an instance where a subgraph of the flat network is given as in Figure 3.2a, and a subgraph of the initial partial network is given in Figure 3.2b. Suppose an optimal routing through D_S assigns commodity 1 to travel along the trajectory formed by the dotted red timed arcs, and commodity 2 to travel along the trajectory formed by the dashed blue timed arc. Observe that the trajectory for commodity 1 includes the timed arc $((v_1,0),(v_2,0))$ which has an overly-optimistic transit time. A DDD algorithm tightens the relaxation and makes this trajectory infeasible by adding the timed node $(v_2,1)$. Note that adding this timed node leads to a new timed copy for arcs v_2v_3 *and* v_2v_4, as shown in Figure 3.2c. However, no timed copy of arc v_2v_4 was used by an infeasible trajectory in the previous lower bound solution. Thus, we would prefer an approach that does not add $((v_2,1),(v_4,3))$ to the partial network.



(a) Base graph D (b) A partial network D_S. (c) Updated partial network.

Figure 3.2

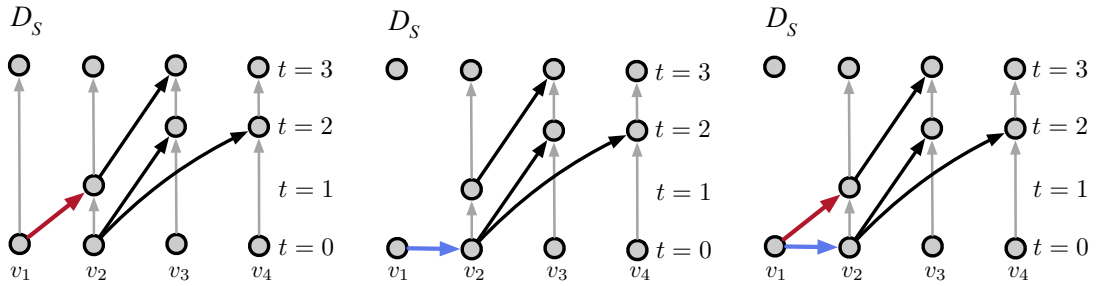
3.1.1 Towards arc-dependent departure times

We want to modify the family of partial networks and corresponding lower bound formulations so that permitting d departure times at a node v does not lead to d timed copies of each arc departing v. In the current DDD approach, each iteration is fully determined by the flat network and a time discretization on the node set, denoted $\mathcal{T}_N = \{\mathcal{T}_v\}_{v \in N}$, where $\mathcal{T}_v \subseteq \{0, 1, \ldots, T\}$. Specifically, the partial network in each iteration has a timed copy of arc vw for each time $t \in \mathcal{T}_v$. Instead, we would like the partial network in each iteration to have timed arcs with departure times according to a time discretization on the arc set, $\mathcal{T}_A = \{\mathcal{T}_{uv}\}_{uv \in A}$, where $\mathcal{T}_{uv} \subseteq \{0, 1, \ldots, T\}$ is the set of departure times for arc uv. Moreover, we need to construct the partial network so that routing the commodities through it still gives a lower bound on the optimal solution. One key consideration is ensuring that each directed flat path in D with transit time at most T is the projection of some trajectory in the partial network. In other words, the partial network *preserves all feasible flat paths*.

A natural first attempt to obtain a sparser partial network is to remove a subset of timed arcs from the standard construction of D_S. We will show that this approach does not guarantee that routing through the resulting partial network gives a lower bound, since it does not preserve all feasible flat paths. Specifically, removing a timed copy of arc vw may cause a flat path containing vw to no longer be the projection of any remaining trajectory. We then show that we can introduce an additional timed arc *entering* v to recover these flat paths. Moreover, we show that for certain instances, the additional timed arc entering v will not lead to additional variables in the corresponding formulation.

Returning to the example in Figure 3.2, suppose we remove timed arc $((v_2, 1), (v_4, 3))$ from Figure 3.2c, as shown in Figure 3.3a. Observe that in the resulting graph, there is no

trajectory with flat path v_1, v_2, v_4 since the copy of arc v_1v_2 arrives at v_2 *later* than the copy of v_2v_4 departs v_2. If we replace timed arc $a = ((v_1, 0), (v_2, 1))$ in Figure 3.3a with $a' = ((v_1, 0), (v_2, 0))$ to form D_S as in Figure 3.3b, then the partial network preserves all feasible flat paths. However, adding a' also returns the red dotted trajectory (Figure 3.2b) to the partial network – precisely the infeasible trajectory which adding timed node $(v_2, 1)$ was meant to remove. Moreover, adding a' made the timed arc $((v_2, 1), (v_3, 3))$ *redundant* for most commodities: any trajectory using $((v_2, 1), (v_3, 3))$ (not originating at v_2) must arrive at v_2 at time 0, and so such a trajectory includes timed arcs $((v_2, 0), (v_2, 1))$ and $((v_2, 1), (v_3, 3))$. This pair of timed arcs can be replaced with $((v_2, 0), (v_3, 2))$ and $((v_3, 2), (v_3, 3))$, without incurring any additional cost.



(a) The v_1v_2 arc is $a = ((v_1, 0), (v_2, 1))$.

(b) The v_1v_2 arc is $a' = ((v_1, 0), (v_2, 0))$.

(c) Both timed arcs are included.

Figure 3.3

To summarize, in the partial network we require a' so that paths containing arc v_2v_4 are preserved, and a so that no copy of v_2v_3 is redundant. Specifically, a is permitted if the next arc taken is v_2v_3, and a' is permitted if the next arc is v_2v_4. Overall, we see that allowing different sets of departure times for arcs in $\delta^+(v_2)$ introduces additional timed copies of arcs in $\delta^-(v_2)$. This seemingly does not fix the density issue, since in order to decrease the number of variables corresponding to timed arcs *leaving* node v, we increased the number of variables corresponding to timed arcs *entering* v.

The core problem with partial networks with many timed arcs is that this often leads to many variables in the corresponding formulation. However, we can avoid introducing variables for each of the timed arcs entering v_2 if each commodity can use at most one arc departing v_2 in a feasible solution. Specifically, if for all $k \in \mathcal{K}$, $|\{v_2v_3, v_2v_4\} \cap D^k| \leq 1$, then there must be a flow variable for at most one of (k, a) and (k, a'). In this case, while the partial network includes additional timed arcs incoming to v_2 this *does not* increase the number of variables for commodity k. Note that if there is some commodity k with

$|\{v_2v_3, v_2v_4\} \cap D^k| = 2$, then we would require variables for both (k, a) and (k, a') if v_2v_3 and v_2v_4 have different departure times.

This observation motivates the partitioning of the arc set presented in Section 3.1.2, where for each $k \in \mathcal{K}$, all arcs in $\delta^+_{D^k}(v)$ are in the same part. We then use this arc partition to build an auxiliary flat network, denoted G, based on the original flat network D. In Section 3.1.3 we model SND-RR on the corresponding full network of G, denoted G_T, and modify the consolidation constraints in order to correct for lost consolidation when additional arc copies are introduced. In Theorem 3.6, we prove that the resulting formulation is equivalent to the original formulation SND-RR(D_T). In Section 3.2, we present a DDD algorithm that uses time-indexed formulations on the auxiliary graph G which often leads to smaller formulations than generic DDD.

3.1.2 Auxiliary graph G

Let $\mathcal{I}$ be an instance of SND-RR with base network $D = (N, A)$, time horizon T, and designated flat network $D^k \subseteq D$ for each commodity $k \in \mathcal{K}$. For each node $v \in N$, we partition the set of arcs in $\delta^+_D(v)$ into arc sets $\mathcal{A}_v = \{A_1, A_2, \ldots, A_r\}$ so that for each commodity $k \in \mathcal{K}$, $\delta^+_{D^k}(v) \subseteq A_i$ for some $A_i \in \mathcal{A}_v$. That is, the arcs departing v that can be used by a fixed commodity in a feasible solution must be contained in a single set in the partition. Using this arc partition, in Section 3.2 we define a DDD approach so that arcs in different sets in $\mathcal{A}_v$ can have different sets of permissible departure times in each iteration of DDD.

Definition 3.1. *A partition $\mathcal{A} = \{\mathcal{A}_v\}_{v \in N}$ of the arcs in $D = (N, A)$ is* valid with respect to the instance $\mathcal{I}$ *if for each $v \in N$,*

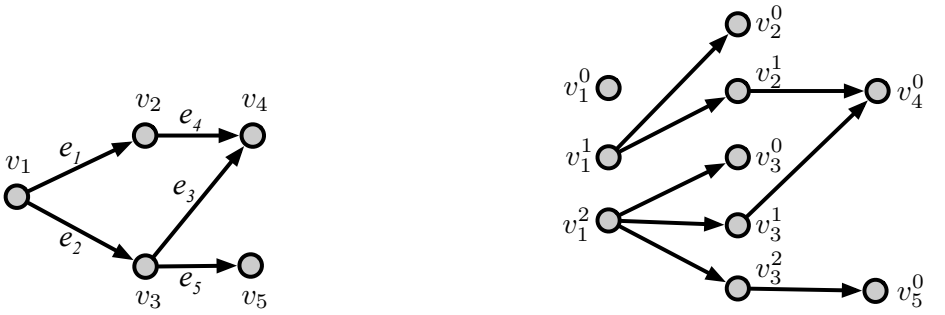
1. *$\mathcal{A}_v = \{A_1, \ldots, A_r\}$ is a partition of $\delta^+_D(v)$, and*
2. *for each commodity $k \in \mathcal{K}$, $\delta^+_{D^k}(v) \subseteq A_i$ for some $A_i \in \mathcal{A}_v$.*

For each node $v \in N$, let $\mathcal{V}(v) := \{v^i : v \in N, i \in \{0, 1, \ldots, |\mathcal{A}_v|\}\}$ denote a set of nodes obtained by creating a copy of v for each set $A_i \in \mathcal{A}_v$, along with a copy v^0 which we refer to as the *terminal copy*. Given a valid partition of the arcs, $\mathcal{A}$, and flat network D, we construct an auxiliary graph where we create a copy of each node v for each part in the partition of $\delta^+(v)$. The arc set is constructed so that there is a copy of each arc vw departing the node copy representing the part containing vw, for each copy of w. This construction is stated more precisely in the following definition.

Definition 3.2. *Given a flat network D and a valid partition of the arc set $\mathcal{A}$, the* auxiliary flat network $G(D, \mathcal{A})$ *has node set*

$$V := \bigcup_{v \in N} \mathcal{V}(v), \text{ and arc set } E = \{v^i w^j : vw \in A_i \in \mathcal{A}, w^j \in \mathcal{V}(w)\}.$$

When the base graph D and arc partition $\mathcal{A}$ are clear from context, we simply write G rather than $G(D, \mathcal{A})$. In Figure 3.4b we provide the auxiliary flat network when D is the flat network in Figure 3.4a and the arc partition is $\mathcal{A} = \{vw\}_{vw \in A}$ (each arc is in its own part). Observe that for each node v in D, the corresponding copy v^0 in G has no outbound arcs, and will only be used by commodities with destination at v.



(a) A flat network D.

(b) $G(D, \mathcal{A})$ when $\mathcal{A} = \{vw\}_{vw \in A}$

Figure 3.4: Construction of the auxiliary flat network

We assign the fixed and variable costs as well as the transit time for each arc copy in E to be the same as its underlying arc in A. The format of the auxiliary network allows us to store the set of available departure times at the node copy v^i for each set of arcs in part $A_i \in \mathcal{A}_v$, rather than forcing all arcs departing a fixed vertex to have the same set of departure times in each iteration of DDD.

3.1.3 Modelling SND-RR on G_T

Recall that in the problem of SND-RR, each commodity has a designated feasible subgraph D^k of D through which it must be routed. That is, the physical route of commodity k must be a path from o_k to d_k in D^k. In order to create a time-indexed formulation where the flat network is the auxiliary graph G, we first define the designated subgraph G^k in G for each commodity $k \in \mathcal{K}$. We then assign the origin and destination for each commodity in G.

Mapping D^k to G^k

Let $\mathcal{A} = \{\mathcal{A}_v\}_{v \in N}$ be the valid arc partition, and for each $k \in \mathcal{K}$ let $D^k = (N^k, A^k)$ denote the designated subgraph for commodity k. For each $k \in \mathcal{K}$, we may assume that $\delta^+_{D^k}(d_k) = \emptyset$, and for all $v \in N^k \setminus d_k$, $\delta^+_{D^k}(v) \neq \emptyset$ as otherwise D^k could be reduced without losing any optimal solutions.

For each set $A_i \in \mathcal{A}$, let $\mathcal{K}(A_i)$ denote the set of commodities that could traverse an arc in A_i in a feasible solution. That is, for each $v \in N$ and each set of arcs $A_i \in \mathcal{A}_v$, let

$$\mathcal{K}(A_i) := \{k \in \mathcal{K} : \emptyset \neq \delta^+_{D^k}(v) \subseteq A_i\}.$$

By definition of a valid partition, for each commodity $k \in \mathcal{K}$, at each node $v \in N^k \setminus \{d_k\}$ there is a single copy of node v in G that has outgoing arcs that could be used by commodity k. More precisely, there is a single set $A_i \in \mathcal{A}_v$ such that $k \in \mathcal{K}(A_i)$. For the destination d_k, we assign the corresponding node to be d_k^0 in G.

For each commodity $k \in \mathcal{K}$ we define the designated subgraph G^k in G as the graph with node set

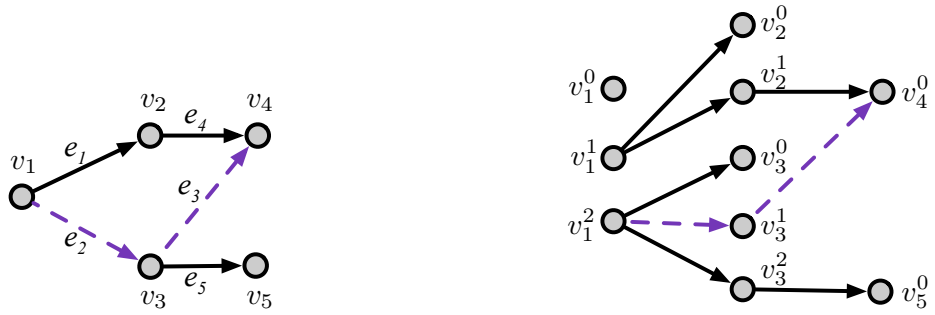
$$V^k := \{v^i : v \in N^k, A_i \in \mathcal{A}_v, k \in \mathcal{K}(A_i)\} \cup \{d_k^0\},$$

and arc set

$$E^k := \{v^i w^j : vw \in A^k, v^i \in V^k, w^j \in V^k\}.$$

Observe that there is a single copy of the origin and destination nodes, o_k and d_k, in V^k. Let o'_k and d_k^0 denote these node copies respectively.

We provide an example of this construction in Figure 3.5, where we continue with the flat network D and auxiliary network $G(D, \mathcal{A})$ from Figure 3.4. In Figure 3.5a we show the subgraph D^k in purple dashed lines, for a commodity with origin v_1 and destination v_4. In Figure 3.5b the subgraph G^k is presented in purple dashed lines, and we see the origin and destination in G^k are $o'_k = v_1^2$ and $d_k^0 = v_4^0$ respectively.



(a) A flat network D and feasible subgraph D^k.

(b) The corresponding subgraph G^k in G.

Figure 3.5: Map from D^k to G^k when D^k is a single path.

Lemma 3.3. *For each commodity $k \in \mathcal{K}$, G^k has the same number of nodes and arcs as D^k.*

Proof. Let $k \in \mathcal{K}$. First, we show that there is a one-to-one correspondence between the nodes in D^k and the nodes in G^k. Let $v \in N^k$, and suppose $v \neq d_k$. Since $\mathcal{A}$ is a valid partition of the arcs and $\delta^+_{D^k}(v) \neq \emptyset$, $k \in \mathcal{K}(A_i)$ for a single set $A_i \in \mathcal{A}_v$. By definition of V^k there is a single copy, v^i, of v in V^k. Suppose instead that $v = d_k$. Then $\delta^+_{D^k}(v) = \emptyset$ and so $k \notin \mathcal{K}(A)$ for all $A \in \mathcal{A}_v$. As a result, the only copy of v in G is d_k^0. Thus, the number of nodes is the same in D^k and G^k.

Since there is a single node in V^k for each node in N^k, there is one copy $v^i w^j$ in E^k of each arc vw in A^k. Therefore the number of arcs in each graph is also the same. □

Furthermore, there is a natural bijection between the set of dipaths in the flat networks D and $G(D, \mathcal{A})$. Let $\mathcal{P}_D$ denote the set of dipaths in D, and let $\mathcal{P}_G$ denote the set of dipaths in G which end at a terminal node. We define a map $\phi : \mathcal{P}_D \to \mathcal{P}_G$ so that for any dipath $P \in \mathcal{P}_D$ with node sequence $\{v_1, v_2, \ldots, v_{j+1}\}$ and arc sequence $\{a_1, a_2, \ldots, a_j\}$, then $\phi(P)$ is the unique dipath in G with the node sequence $\{v_1^{i_1}, v_2^{i_2}, \ldots, v_j^{i_j}, v_{j+1}^0\}$, where for each $\ell \in [j]$, a_l is in $A_{i_l} \in \mathcal{A}$. We think of ϕ as both as a map of the vertices and arcs of a trajectory. Let ϕ^{-1} denote the inverse of ϕ. That is, ϕ^{-1} takes a path P in G and projects each arc (node) copy in G down to its corresponding arc (node) in D.

Fact 3.4. *$|\mathcal{P}_D| = |\mathcal{P}_G|$, and ϕ is a bijection between $\mathcal{P}_D$ and $\mathcal{P}_G$.*

Given a time horizon T, let $G_T = (V_T, E_T \cup F_T)$, denote the full network with respect to G and T, where E_T denotes the movement arcs and F_T denotes the holdover arcs. Let G_T^k denote the full network corresponding to G_k. Fact 3.4 naturally extends to the following

analogous statement for D_T and G_T. Let $\mathcal{Q}_D$ denote the set of trajectories in D_T, and let $\mathcal{Q}_G$ denote the set of trajectories in G_T that end at a timed terminal node.

We define the map $\phi_T : \mathcal{Q}_D \to \mathcal{Q}_G$ as follows (again, we will think of ϕ_T as mapping both timed arcs and timed nodes). Let $Q \in \mathcal{Q}_D$ be a trajectory in D_T and let P be the corresponding flat path in D. Note that any trajectory in D_T is is fully determined by its physical path in D along with a departure time for each arc in the path. We define $\phi_T(Q)$ as the trajectory in G_T defined by the path $\phi(P)$ along with the departure times of Q. Let ϕ_T^{-1} denote the inverse of ϕ_T. That is, ϕ_T^{-1} takes a trajectory Q in G_T and projects each timed arc (timed node) copy in G_T down to its corresponding timed arc (timed node) in D_T.

Fact 3.5. *$|\mathcal{Q}_D| = |\mathcal{Q}_G|$, and ϕ_T is a bijection between $\mathcal{Q}_D$ and $\mathcal{Q}_G$.*

These bijections allow us to argue that the following formulation for SND-RR is correct.

The formulation SND-RR(G_T)

In the definition of G, there are multiple copies of the same arc $vw \in A$. Suppose two commodities k and k' traverse the arc vw at the same time, but then travel along arcs departing w in different parts according to the partition $\mathcal{A}$. As a result, while it was possible for each commodity to traverse the same timed copy of vw in D_T, it is *no longer possible* in G_T. To capture the ability to consolidate flow in G_T, we define subsets of timed arcs in G_T which have the same departure time and underlying arc in D (as opposed to G). That is, for each timed arc $a = ((v,t),(w,t')) \in A_T$, we define the set

$$\mathcal{E}_T(a) := \{((v^i,t),(w^j,t')) : vw \in A_i \in \mathcal{A}_v, w^j \in \mathcal{V}(w)\}.$$

We now present a formulation for solving instances of SND-RR, denoted SND-RR(G_T), which is defined on the time-expanded network G_T. We will continue to use a to denote timed arcs in D_T, and will use e to denote timed arcs in G_T. Note that there are a few key differences between SND-RR(D_T) and SND-RR(G_T). First, G_T is a larger graph, even in the case where $\mathcal{A}$ is the trivial partition, $\{\delta^+(v)\}_{v \in N}$, in which case G_T is roughly double the size of D_T (there are two copies of each node – the copy with departing arcs, and the terminal copy). We also restrict commodities to the designated subgraphs G_T^k in G_T rather than in D_T. Finally, we capture the consolidation onto different arc copies through

modifying constraint (3.2) for all $a \in A_T$ by summing over each timed arc copy in $\mathcal{E}_T(a)$.

$$\min \quad \sum_{a \in A_T} f_a y_a + \sum_{k \in \mathcal{K}} \sum_{e \in E_T} c_e^k q_k x_e^k \tag{SND-RR(G_T)}$$

$$\text{s.t.} \quad x^k(\delta^+_{G_T^k}(v,t)) - x^k(\delta^-_{G_T^k}(v,t)) = \begin{cases} 1 & (v,t) = (o'_k, r_k) \\ -1 & (v,t) = (d_k^0, l_k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in V_T^k \tag{3.1}$$

$$\sum_{k \in \mathcal{K}} \sum_{e \in \mathcal{E}_T(a)} q_k x_e^k \leq u_a y_a \quad \forall a \in A_T \tag{3.2}$$

$$x_e^k \in \{0,1\} \quad \forall k \in \mathcal{K}, e \in E_T^k \cup F_T^k \tag{3.3}$$

$$y_a \in \mathbb{N}_{\geq 0} \quad \forall a \in A_T \tag{3.4}$$

However, despite working with a larger graph G_T, the number of variables and constraints are the same in SND-RR(D_T) and SND-RR(G_T).

Theorem 3.6. *SND-RR(D_T) and SND-RR(G_T) have the same number of variables and constraints, and are equivalent.*

Proof. By Lemma 3.3, for each $k \in \mathcal{K}$, D^k and G^k have the same number of nodes and arcs. This implies that D_T^k and G_T^k also have the same number of timed nodes and timed arcs since the transit time of an arc in G is the same as the corresponding arc in D. Thus, the number of variables and constraints in SND-RR(D_T) and SND-RR(G_T) is the same.

We now prove that there is a cost-preserving bijection between feasible solutions in SND-RR(D_T) and feasible solutions in SND-RR(G_T). Let $(\bar{x}, \bar{y})$ be a feasible solution in SND-RR(D_T) with corresponding trajectories $\mathcal{Q} = \{Q_k\}_{k \in \mathcal{K}}$ in D_T and paths $\mathcal{P} = \{P_k\}_{k \in \mathcal{K}}$ in D. For each $k \in \mathcal{K}$, consider the set of trajectories $\phi_T(\mathcal{Q}) = \{\phi_T(Q_k)\}_{k \in \mathcal{K}}$. Since the physical path of Q_k is $\phi(P_k)$, and $P_k \subseteq D^k$, it follows from the definition of G^k that $\phi(P_k) \subseteq G^k$. Additionally, $\phi(P_k)$ has origin o'_k and destination d_k^0. Finally, since the departure times are the same for arcs in P_k and their corresponding arc copies in $\phi(P_k)$, it follows that $\phi(Q_k)$ is feasible for commodity k in SND-RR(G_T).

It remains to argue that the cost of $\phi_T(\mathcal{Q})$ in SND-RR(G_T) is the same as the cost of $\mathcal{Q}$ in SND-RR(D_T). Since arc copies in G are assigned the same costs as the underlying arcs in D and $\phi(P)$ simply assigns each arc in P to one of its copies in G, the variable cost for each trajectory is the same for $\mathcal{Q}$ and $\phi_T(\mathcal{Q})$. We now consider the fixed cost for timed arc $a \in A_T$. Since $\mathcal{Q}$ and $\phi_T(\mathcal{Q})$ have the same set of departure times and ϕ maps each

arc in the flat path to one of its copies in G, it follows that the same set of commodities k appear in the capacity constraint for timed arc a in SND-RR(D_T) and SND-RR(G_T). Therefore the fixed cost of $\phi_T(\mathcal{Q})$ in SND-RR(G_T) is also the same as the fixed cost of $\mathcal{Q}$ in SND-RR(D_T). The reverse direction is analogous. □

Thus, in order to solve SND-RR(D_T), we can instead solve SND-RR(G_T). As previously mentioned, moving to SND-RR(G_T) has no advantage over SND-RR(D_T) if we are solving the formulations directly with a MIP solver since they are isomorphic. However, we will show that the DDD paradigm can be improved when applied to the base graph G rather than D. Note that even if $D^k = D$ for some (or all) $k \in \mathcal{K}$, the auxiliary network still splits each node in D into two nodes (one for departing arcs, and the other denoting the terminal copy).

3.2 Arc-based DDD approach

We now define a new DDD algorithm based on a lower bound formulation for SND-RR(G_T). We describe the lower bound formulation, upper bound, and refinement processes required for a complete DDD algorithm. We will refer to the DDD algorithm based on the auxiliary network as an *arc-based* DDD algorithm, and the original approach as a *node-based* DDD algorithm. The arc-based DDD approach is not simply the standard DDD algorithm applied to this new auxiliary network. Instead, the approach must be modified to ensure costs continue to capture all possible consolidation opportunities.

The advantage of using the auxiliary network is that in the corresponding DDD algorithm, each set of arcs in the same part of the arc partition has its own set of departure times. As a result, in each iteration fewer variables and constraints need to be added in order to improve the current network while maintaining a guaranteed lower bound. Specifically, when lengthening a short arc $((v,t),(w,t'))$, we can now add the departure time $t+\tau_{vw}$ to a subset of the arcs departing w, rather than the entire set. In Section 3.3, we highlight two particular applications where the arc-based approach has the potential to generate smaller iterations than the node-based approach.

In this section we assume $\mathcal{A}$ is a valid partition of the arc set A, and $G = G(D,\mathcal{A})$ is the corresponding auxiliary network. Furthermore, each commodity $k \in \mathcal{K}$ has a designated network $D^k \subseteq D$, and designated network $G^k \subseteq G$. Additionally, the origin and destination of commodity k in G are denoted o'_k and d^0_k respectively.

3.2.1 Lower bound model

Just as in the case of applying DDD to the base graph $D = (N, A)$, we require restrictions on the structure of a partial network of $G = (V, E)$, denoted $G_S = (V_S, E_S \cup F_S)$, and also require an accompanying formulation. We first begin with restrictions on G_S.

Properties to guarantee a lower bound

We will prove that when a partial network $G_S = (V_S, E_S \cup F_S)$ satisfies the following two properties, the optimal value of the corresponding formulation SND-RR(G_S) defined in Section 3.2.1 gives a lower bound on the value of SND-RR(G_T). These properties are analogous to properties (P1) and (P2) introduced in Section 2.2. Recall that for each $v \in N$, $\mathcal{V}(v)$ denotes the set of copies of v in G.

(P1′) **Timed nodes**:
For all $k \in \mathcal{K}$, $(o'_k, r_k) \in V_S$ and $(d^0_k, l_k) \in V_S$;
For all $v \in N$, $(v^0, T) \in V_S$ and $(u, 0) \in V_S$ for all $u \in \mathcal{V}(v)$;

(P2′) **Arc copies**:
For all $vw \in E$, for all $(v, t) \in V_S$ with $t + \tau_{vw} \leq T$, we have $((v, t), (w, t')) \in E_S$ where $t' = \max\{r : r \leq t + \tau_{vw}, (w, r) \in V_S\}$;

We define F_S to be the set of holdover arcs connecting V_S. Given $G_S = (V_S, E_S \cup F_S)$, for each $k \in \mathcal{K}$ we obtain $G^k_S = (V^k_S, E^k_S \cup F^k_S)$ where $V^k_S = \{(v, t) \in V_S : v \in V^k\}$, $E^k_S = \{((v, t), (w, t')) \in E_S : vw \in E^k\}$, and F^k_S is the corresponding set of holdover arcs.

Notation

Let $D_S(G_S) = (N_S, A_S \cup H_S)$ denote the partial network with respect to D with timed nodes

$$N_S = \{(v, t) : (u, t) \in V_S \text{ for some } u \in \mathcal{V}(v)\},$$

and A_S constructed according to $(P2)$. Similar to the definition of $\mathcal{E}_T$, for each timed arc $a = ((v, t), (w, t'))$ in A_S, let $\mathcal{E}_S(a)$ denote the set of timed arc copies of vw with departure time t. That is,

$$\mathcal{E}_S(a) := \{((v^i, t), (w^j, t'')) \in E_S : w^j \in \mathcal{V}(w), vw \in A_i \in \mathcal{A}_v\}.$$

We may have $e = ((v^i, t), (w^j, t''))$ where $t'' \neq t'$ due to rounding down transit times in G_S. Note that each timed arc $e \in E_S$ is in a set $\mathcal{E}_S(a)$ for exactly one timed arc $a \in A_S$.

Lower bound formulation

We now state the lower bound formulation defined on valid partial auxiliary networks G_S. The set A_S in the formulation is the set of timed movement arcs in $D_S(G_S)$. In the following formulation, for $e = ((v,t),(w,t')) \in E_S$, τ_e denotes the transit time of arc vw rather than $t' - t$.

$$\min \quad \sum_{a \in A_S} f_a y_a + \sum_{k \in \mathcal{K}} \sum_{e \in E_S} c_e^k q_k x_e^k \tag{SND-RR(G_S))}$$

$$\text{s.t.} \quad x^k(\delta^+_{G_S^k}(v,t)) - x^k(\delta^-_{G_S^k}(v,t)) = \begin{cases} 1 & (v,t) = (o'_k, r_k) \\ -1 & (v,t) = (d_k^0, l_k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in V_S^k \tag{3.5}$$

$$\sum_{k \in \mathcal{K}} \sum_{e \in \mathcal{E}_S(a)} q_k x_e^k \leq u_a y_a \quad \forall a \in A_S \tag{3.6}$$

$$\sum_{e \in E_S^k} \tau_e x_e^k \leq l_k - r_k \quad \forall k \in \mathcal{K}. \tag{3.7}$$

$$x_e^k \in \{0,1\} \quad \forall k \in \mathcal{K}, e \in E_S^k \cup F_S^k \tag{3.8}$$

$$y_a \in \mathbb{N}_{\geq 0} \quad \forall a \in A_S \tag{3.9}$$

Constraint (3.5) ensures each commodity k is assigned a feasible trajectory in G_S^k. Constraint (3.6) is similar to the capacity constraint (3.2) in SND-RR(G_T), with the additional relaxation that flow can consolidate onto a common truck if it *departs* along some copy of the base arc in D at the same time. Observe that when $G_S = G_T$, it follows that SND-RR(G_S) is the same as SND-RR(G_T).

We now prove that properties $(P1')$ and $(P2')$ are sufficient to ensure that SND-RR(G_S) is a relaxation of SND-RR(G_T). The proof of this result is similar to the proof of Theorem 2 [2], when given the base graph G instead of D. However, careful attention is taken to track the consolidation of flow, which is pointed out near the end of the proof. In the following proof, while we are working with nodes in V_T and V_S, we drop the superscripts denoting the copy of the node in D for ease of notation until the end of the proof.

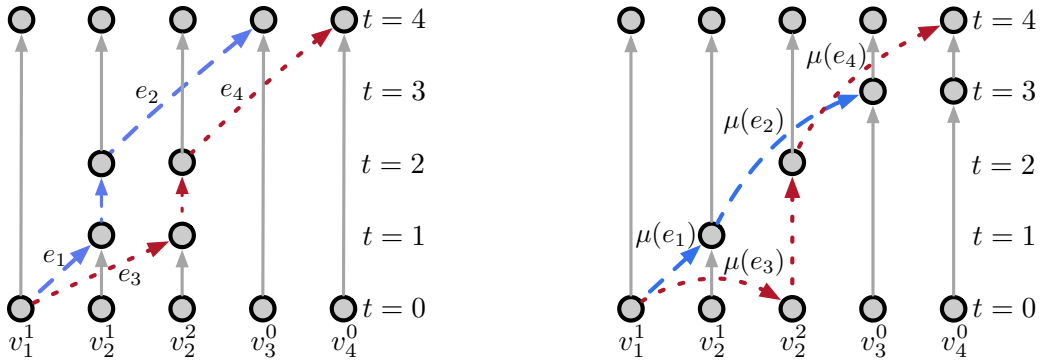
Theorem 3.7. *When G_S satisfies properties $(P1')$ and $(P2')$, the optimal value of (SND-RR(G_S)) provides a lower bound on the optimal value of (SND-RR(G_T)).*

Proof. Let $(\bar{x}, \bar{y})$ be a feasible solution to (SND-RR(G_T)) with cost C, and let $\mathcal{Q} = \{Q_k\}_{k \in \mathcal{K}}$ denote the corresponding set of trajectories. Let $\mu : E_T \to E_S$ be the map

defined so that for each timed arc $e \in E_T$,

$$e = ((v,t),(w,t')) \quad \rightarrow \quad \mu(e) = ((v,\hat{t}),(w,\hat{t}')), \tag{3.10}$$

where $\hat{t} = \max\{s : s \leq t, (v,s) \in V_S\}$, and $\hat{t}' = \max\{s : s \leq \hat{t} + \tau_{vw}, (v,s) \in V_S\}$. Observe that $\hat{t}'$ is dependent on $\hat{t}$ rather than t'. Furthermore, μ is well-defined since $(P1')$ dictates that there is a timed node for each non-terminal node in V with time 0, and $(P2')$ ensures that there is a (unique) copy of each arc in E departing the selected timed node that underestimates the correct transit time. Examples of the map μ are given in Figure 3.6, where Figure 3.6a shows the original timed arcs on a subgraph of G_T, and Figure 3.6b shows the resulting timed arcs after applying μ, given a partial auxiliary network with the timed node set as shown.



(a) Two trajectories in G_T.

(b) Corresponding trajectories in G_S.

Figure 3.6

We obtain trajectories $\hat{\mathcal{Q}} = \{\hat{Q}_k\}_{k \in \mathcal{K}}$ by mapping each movement arc $e \in E_T$ to $\mu(e) \in E_S$, and forming trajectories by adding holdover arcs. Figure 3.6 shows two examples of this map from trajectories in G_T to G_S. We now describe this process more precisely, and prove that the resulting trajectories are well-defined and feasible in G_S. Let Q_k be the trajectory induced by $\bar{x}$ for commodity k. Then the set of ordered movement arcs in Q_k is $\{e_1, e_2, \ldots, e_{q_k}\}$, where $e_i = ((v_i, t_i^{out}), (v_{i+1}, t_{i+1}^{in}))$ for each $i \in [q_k - 1]$, and since Q_k is feasible, the following statements are true:

(S1) $v_1 = o'_k$, and $t_1^{out} \geq r_k$
(S2) $v_{q_k} = d_k^0$, and $t_{q_k}^{in} \leq l_k$
(S3) $t_{i+1}^{in} = t_i^{out} + \tau_{v_i v_{i+1}}$ for all $i \in [q_k - 1]$
(S4) $t_i^{in} \leq t_i^{out}$ for all $i \in \{2, 3, \ldots, q_k - 1\}$

We apply the map μ to each movement arc and obtain the ordered set of movement arcs $\{\mu(e_1), \mu(e_2), \ldots, \mu(e_{q_k})\}$ in E_S, where $\mu(e_i) = ((v_i, \hat{t}_i^{out}), (v_{i+1}, \hat{t}_{i+1}^{in}))$ for each $i \in [q_k - 1]$. By an analogous argument to the proof of Claim 1 in the proof of Theorem 2.2, $\hat{t}_i^{in} \leq \hat{t}_i^{out}$ for all $i \in \{2, 3, \ldots, q_k - 1\}$. As a result, we can add holdover arcs to form trajectories for each $k \in \mathcal{K}$. Observe that each $\hat{Q}_k$ is contained in G_S^k, and by property $(P1')$ along with (S1) and (S2), we see that $\hat{t}_1^{out} \geq r_k$ and $\hat{t}_{q_k} \leq l_k$. Therefore $\hat{Q}_k$ is feasible in G_S for commodity k.

It remains to show that the cost of $\hat{\mathcal{Q}}$ in G_S is at most the cost of $\mathcal{Q}$. Since the underlying paths in the flat network for $\hat{\mathcal{Q}}$ and $\mathcal{Q}$ are the same for each commodity, the variable cost of $\hat{\mathcal{Q}}$ and $\mathcal{Q}$ is the same. Additionally, all trajectories in $\hat{\mathcal{Q}}$ have flat paths with feasible total transit time (at most $l_k - r_k$ for each $k \in \mathcal{K}$).

We now prove that the fixed cost of $\hat{\mathcal{Q}}$ is at most the fixed cost of $\mathcal{Q}$. Observe that and any pair of commodities k, k' traversing the same timed arc $e \in E_T$ now traverse the same timed arc $\mu(e) \in E_S$. It remains to prove that if two distinct timed arcs e and e' in E_T are in $\mathcal{E}_T(a)$ for some timed arc $a \in A_T$, then $\mu(e)$ and $\mu(e')$ are in $\mathcal{E}_S(a')$ for some $a' \in A_S$.

Suppose e and e' are in $\mathcal{E}_T(a)$ for some $a = ((v, t), (w, t')) \in A_T$, and $vw \in A_i \in \mathcal{A}_v$. Then $e = ((v^i, t), (w^j, t'))$ and $e' = ((v^i, t), (w^p, t''))$, where $w^j, w^p \in \mathcal{V}(w)$. It follows that $\mu(e) = ((v^i, \hat{t}), (w^p, \hat{t}'))$ and $\mu(e') = ((v^i, \hat{t}), (w^p, \hat{t}''))$. Therefore, $\mu(e)$ and $\mu(e')$ are in the set $\mathcal{E}_S(a')$, where $a' = ((v, \hat{t}), (w, \hat{t} + \tau_{vw}))$. An example is given in Figure 3.6 for the timed arcs e_1 and e_3. Observe that e_1 and e_3 are in the set $\mathcal{E}_T(a)$ for $a = ((v_1, 0), (v_2, 1)) \in A_T$, and $\mu(e_1)$ and $\mu(e_3)$ are in the set $\mathcal{E}_T(a')$ where $a' = ((v_1, 0), (v_2, 1))$. Thus, the fixed cost of $\hat{\mathcal{Q}}$ is at most the fixed cost of $\mathcal{Q}$. □

Observe that when $G_S = G_T$, SND-RR(G_S) is equal to SND-RR(G_T). To fully define the DDD approach, it remains to outline the upper bound/termination procedure and the refinement procedure.

3.2.2 Upper bound and refinement

Let $(\hat{x}, \hat{y})$ be an optimal solution to SND-RR(G_S) with corresponding trajectories $\mathcal{Q}$ in G_S. In each iteration, we provide a feasible solution to SND-RR(D_T) as well as a set of timed nodes to add to G_S if the feasible solution is not optimal. As in the case of the original node-based DDD approach, the generation of the feasible solution in each iteration is found by solving a continuous formulation whose input is defined by the solution $(\hat{x}, \hat{y})$ to the lower bound model. Both the arc-based DDD and node-based DDD approaches solve the same continuous formulation, LP-UB($\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F$), and the methods only diverge in the

definition of the inputs to the formulation. We want to define the input $(\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F)$ so that when LP-UB$(\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F)$ has an optimal solution $(\bar{t}, \bar{\delta})$ with value 0, then $(\bar{t}, \mathcal{P})$ defines an optimal solution to SND-RR(D_T).

The first difference from the node-based approach is that the physical paths for $\mathcal{Q}$ are in G rather than D. In order to apply the continuous formulation from the node-based approach, the paths in G are projected down to the original flat network D via the map ϕ. The second difference is in the definition of the sets in $\mathcal{J} = \{J_{vw}\}_{vw \in A}$, which capture the savings in fixed costs due to consolidation. In the original node-based DDD approach, the set of commodities that contribute to the same (fixed charge) capacity constraint of timed arc $a \in A_S$ in SND-RR(D_S) are all commodities that traverse a. In SND-RR(G_S), commodities contribute to the same capacity constraint for timed arc $a = ((v,t),(w,t')) \in D_S(G_S)$ if they depart along any *copy* of arc vw at the same time. The construction of the input $(\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F)$ in each iteration is presented in Algorithm 3.

Algorithm 3: `UB-input`$(D, G_S, D_S(G_S), (\hat{x}, \hat{y}))$

Input: flat network $D = (N, A)$, partial (auxiliary) network $G_S = (V_S, E_S \cup F_S)$, corresponding partial network $D_S(G_S) = (N_S, A_S \cup H_S)$, and optimal solution $(\hat{x}, \hat{y})$ to SND-RR(G_S)

Let $\mathcal{Q} = \{Q_k\}_{k \in \mathcal{K}}$ denote the set of trajectories in G_S given by $\hat{x}$
$\mathcal{P} \leftarrow \{P_k\}_{k \in \mathcal{K}}$, where $P_k = \phi(P'_k) \in D$ and P'_k is the underlying path of Q_k in G
for $vw \in A$ **do**
  $J_{vw} := \{(k_1, k_2) : \exists a = ((v,t),(w,t')) \in A_S, \exists f_1, f_2 \in \mathcal{E}_S(a)$ such that $\hat{x}^{k_1}_{f_1} = \hat{x}^{k_2}_{f_2}\}$.
$\mathcal{J} \leftarrow \{J_{vw}\}_{vw \in A}$
$\mathcal{K}^F := \{k \in \mathcal{K} : \forall f = ((v,t),(w,t')) \in A_S$ such that $\hat{x}^k_f > 0$, $t' = t + \tau_{vw}\}$
for $k \in \mathcal{K}^F$ **do**
  $\mathcal{T}_k = \{\bar{t}^k_{v_p}\}_{p \in [|P_k|-1]}$, where $\bar{t}^k_{v_p} := \{t : \exists u \in \mathcal{V}(v_p), \hat{x}^k(\delta^+_{E_S}(u,t)) > 0\}$. ie, the time commodity k leaves a copy of v_p in $\hat{x}$.
$\mathcal{T}^F \leftarrow \{\mathcal{T}_k\}_{k \in \mathcal{K}^F}$
return $(\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F)$

As in the case of the node-based approach, there is an optimal solution, $(\bar{t}, \bar{\delta})$, to LP-UB$(\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F)$ in each iteration and so a feasible solution $(\bar{x}, \bar{y})$ to SND-RR(D_T) is obtained in each iteration from $\bar{t}$ and $\mathcal{P}$. When $\bar{\delta} = \mathbf{0}$, we see that $(\bar{x}, \bar{y})$ is an optimal solution for SND-RR(D_T), since it has the same variable and fixed cost as $(\hat{x}, \hat{y})$. When the optimal value to LP-UB$(\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F)$ is not equal to 0, we need to refine the partial

network G_S. The refinement process is again analogous to the node-based approach. When the optimal $(\bar{t}, \bar{\delta})$ solution has non-zero value, $\bar{\delta}$ defines a set of commodities, $\mathcal{C}$, with infeasible trajectories. This process is presented in Algorithm 4, and we formally state this result in Theorem 3.8.

Algorithm 4: $\texttt{UB}(D, G_S, D_S(G_S), (\hat{x}, \hat{y}))$

Input: flat network $D = (N, A)$, partial (auxiliary) network G_S, partial network $D_S(G_S)$, and optimal solution $(\hat{x}, \hat{y})$ to SND-RR(G_S)

1 $(\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F) \leftarrow \texttt{UB-input}(D, G_S, D_S(G_S), (\hat{x}, \hat{y}))$

2 Let $(\bar{t}, \bar{\delta})$ denote an optimal solution to LP-UB($\mathcal{P}, \mathcal{J}, \mathcal{K}^F, \mathcal{T}^F$)

3 Let $\bar{Q}$ be the trajectories in G_T defined by $\mathcal{P}, \bar{t}$ and let $(\bar{x}, \bar{y})$ be the corresponding solution to SND-RR(G_T).

4 $\mathcal{C} = \{k_1 \in \mathcal{K} \setminus \mathcal{K}^F : \exists vw \in A, k_2 \in \mathcal{K}, \bar{\delta}_{vw}^{k_1 k_2} > 0\}$

5 **return** $(\bar{x}, \bar{y}), \mathcal{C}$

Theorem 3.8. *Given an optimal solution $(\hat{x}, \hat{y})$ to SND-RR(G_S) with corresponding trajectories $\mathcal{Q}$ in G_S, Algorithm 4 either finds an optimal solution to SND-RR(D_T), or provides a feasible solution to SND-RR(D_T) along with a nonempty set $\mathcal{C}$ of commodities with infeasible trajectories in G_S.*

Similar to the node-based approach, in the refinement step, a timed node is added to V_S to lengthen the earliest-departing short timed arc in each trajectory Q_k where $k \in \mathcal{C}$. Since the timed nodes are added to V_S rather than N_S, the increase in size of the subsequent lower bound formulation is typically less than the increase in size if the timed nodes were added to N_S.

3.2.3 Pseudocode for the arc-based DDD approach

The following algorithm states the overall arc-based DDD approach for solving SND-RR.

Algorithm 5: `SND-RR-arc-disc`$(D, \mathcal{K})$

Input: Base network $D = (N, A)$, commodity set $\mathcal{K}$ with time horizon T

1 **Initialization**
2 Let $G = (V, E)$ be the auxiliary network of D
3 $V_S \leftarrow V_0$, where V_0 is the minimal set of timed nodes satisfying $(P1')$
4 **while** *not solved* **do**
5 **Lower bound**
6 Construct G_S given updated V_S using property $(P2')$.
7 Solve SND-RR(G_S) and obtain a solution $(\hat{x}, \hat{y})$ to SND-RR(G_S).
8 **Upper bound/termination**
9 Solve `UB`$(D, G_S, D_S(G_S), (\hat{x}, \hat{y}))$ for the set $\mathcal{C}$ and upper bound solution $(\bar{x}, \bar{y})$.
10 **if** *the cost of* $(\bar{x}, \bar{y})$ *is equal to the cost of* $(\hat{x}, \hat{y})$ **then**
11 Stop. $(\bar{x}, \bar{y})$ is an optimal solution to SND(D_T).
12 **Refinement**
13 For each $k \in \mathcal{C}$, lengthen the short timed arc $((v, t), (w, t')) \in Q_k$ with the earliest departure time (t), by adding $(w, t + \tau_{vw})$ to V_S.

Theorem 3.9. *The arc-based DDD algorithm, Algorithm 5, terminates with an optimal solution.*

Proof. The algorithm terminates when line 11 is executed. Since the cost of $(\hat{x}, \hat{y})$ gives a lower bound on the optimal value of SND-RR(G_T) (Theorem 3.7) and since $(\bar{x}, \bar{y})$ is feasible for SND-RR(G_T) (Theorem 3.8), if line 11 is executed then $(\bar{x}, \bar{y})$ is an optimal solution.

It remains to bound the number of iterations until line 11 is reached. If in an iteration line 11 is not reached, then the set $\mathcal{C}$ is nonempty (Theorem 3.8) and consists of commodities with trajectories in G_S with at least one short timed arc. As a result, in each iteration at least one timed (auxiliary) node is added to V_S. Thus, the algorithm terminates with an optimal solution in at most $|V| \cdot T$ iterations, where $|V| \leq 2|A|$. □

3.3 Applications

The arc-based DDD approach has greater potential to outperform the traditional node-based approach when the instance admits a fine arc partition. In this section, we examine how both individual route restrictions on commodities, and global restrictions determined

by graph structures, permit fine arc partitions. In particular, we examine settings where paths are designated in the base graph for each commodity, and where the base graph has a region-based structure. We also present an algorithm for determining arc partitions in arbitrary digraphs.

3.3.1 SND with designated paths

First, we consider the problem of SND-RR where the designated flat network D^k for commodity $k \in \mathcal{K}$ is a single path P_k in D. This structure allows each arc to have an independent set of departure times in each iteration, as stated in the following Theorem.

Theorem 3.10. *For all instances $\mathcal{I}$ of SND-RR where D^k is a single path for each commodity $k \in \mathcal{K}$, the arc partition $\mathcal{A}$ where each set $A \in \mathcal{A}$ is a single arc is valid for $\mathcal{I}$.*

Proof. Since each commodity has a designated path, for each node $v \in N$ and each $k \in \mathcal{K}$, $|\delta^+_{D^k}(v) \cap \delta^+_D(v)| \leq 1$. Thus, *any* partition of the arc set A is valid. □

The partition of the arc set into singletons is the ideal scenario, since it allows each arc to have its own set of departure times in each iteration of arc-based DDD. The advantage of this fact can be seen already in the initial partial network. Consider the graph structure in Figure 3.7, presented previously in Section 3.1. For each $i \in [m]$, suppose there is a commodity k with origin v, destination v_i, and release time i. We previously showed that the initial partial network for the node-based DDD approach would have a copy of node v at all times in $[m]$. As a result, D_0 would have a copy of each arc for each departure time in $[m]$. Thus, we would have $A_0 \approx A_T$, and so the size of SND-RR(D_S) is approximately the same as SND-RR(D_T). However, in the arc-based approach, G_0 only requires a single copy of arc vv_i departing v at time i for each $i \in [m]$. Thus, the resulting formulation SND-RR(G_S) is a factor m smaller than SND-RR(D_S).

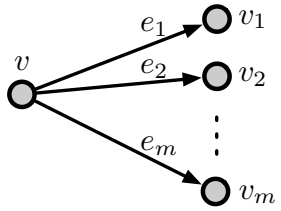


Figure 3.7

In Section 3.4, we compare the performance of the arc-based and node-based DDD algorithms on a class SND-RR instances where each commodity has a designated path.

3.3.2 Hub-and-spoke networks

We now consider hub-and-spoke, a popular region-based construction used in fulfillment and airline networks including FedEx and UPS [4]. In a hub-and-spoke network, locations are divided into regions where each region is represented by a hub. Packages that have origins and destinations in two separate regions must travel between regions via the hubs. In Figure 3.8, hubs are indicated as gray squares and only arcs for inter-region shipments are shown.

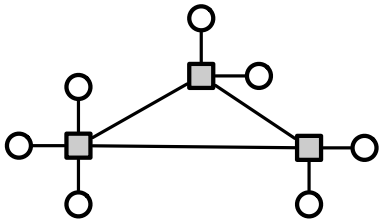


Figure 3.8: Hub-and-spoke network

In fulfillment networks, it is often the case that packages have cut-off times in order to reach their destination warehouse for the next day. As a result, many intra-region arcs leaving a hub will only require a subset of the departure times, namely, those departure times closely preceding the cut-off time. This points to the utility of allowing different departure times for arcs leaving hub nodes. To demonstrate that the arc-based DDD approach has advantages over the traditional node-based DDD approach, we generate a valid partition $\mathcal{A}$ of the arc set. First, we require additional notation.

In a hub-and-spoke network, the node set N is partitioned into a set of regions, $\mathcal{R} = \{R_1, R_2, \ldots, R_\ell\}$, with corresponding hubs $\mathcal{H} = \{h_1, h_2, \ldots, h_\ell\}$ where hub h_i is in region R_i for each $i \in [\ell]$. We refer to arcs as *regional* if they connect two vertices within the same region, and *national* if they connect two nodes in different regions (which must be hub nodes). Let RA and NA denote the set of regional and national arcs in A respectively. Let $\mathcal{R}(v)$ denote the region of the node v, and let $\mathcal{H}(v)$ denote the hub node of the region containing v.

For each commodity $k \in \mathcal{K}$, there is a natural restriction of the flat network when variable and fixed costs are nonnegative. Most notably, commodity k will only potentially use regional arcs in $\mathcal{R}(o_k)$ and $\mathcal{R}(d_k)$. Furthermore, commodity k will not use any national arc departing $\mathcal{H}(d_k)$ or entering $\mathcal{H}(o_k)$, or any regional arc departing $\mathcal{H}(o_k)$. Let $D^k \subseteq D$ consist of the regional arcs in $\mathcal{R}(o_k)$ and $\mathcal{R}(d_k)$, as well as all national arcs except those in $\delta_D^+(\mathcal{H}(d_k))$ and $\delta_D^-(\mathcal{H}(o_k))$. As discussed, restricting the route of commodity k to D^k does not increase the cost of an optimal solution.

This definition of D^k for each $k \in \mathcal{K}$ gives a natural partitioning of the arcs in $\delta_D^+(h)$ for each hub $h \in \mathcal{H}$.

Theorem 3.11. *Let $\mathcal{I} = (D, \mathcal{K})$ be an instance of SND-RR where $D = (N, A)$ has a hub-and-spoke network structure with hubs $\mathcal{H}$, national arcs NA, and regional arcs RA. Let $\mathcal{A} = \{\mathcal{A}_v\}_{v \in N}$ be the arc partition where*

1. *$\mathcal{A}_h = \{A_1^h, A_2^h\}$ where $A_1^h = \delta_D^+(h) \cap RA$ and $A_2^h = \delta_D^+(h) \cap NA$ for all $h \in \mathcal{H}$*
2. *$\mathcal{A}_v = \{\delta_D^+(v)\}$ for all $v \in N \setminus \mathcal{H}$,*

Then $\mathcal{A}$ is a valid partition of A for $\mathcal{I}$.

Proof. Let $k \in \mathcal{K}$, and consider a hub $h \in \mathcal{H}$. We may assume that D^k is minimal in the sense that all arcs that cannot be used by commodity k in an optimal solution have been removed. First suppose $\mathcal{H}(o_k) = \mathcal{H}(d_k) = h$. In this case, commodity k will not use any national arcs departing h, and so $\delta_{D^k}^+(h) \subseteq A_1^h$ and $\delta_{D^k}^+(h') = \emptyset$ for all $h' \in \mathcal{H} \setminus \{h\}$.

Suppose instead that $\mathcal{H}(o_k) \neq \mathcal{H}(d_k)$. For any hub $h \in \mathcal{H} \setminus \mathcal{H}(d_k)$, $\delta_{D^k}^+(h) \subseteq A_2^h$ since commodity k will not use any regional arc departing h. If $h = \mathcal{H}(d_k)$, then similarly commodity k will not use any national arc departing h, so $\delta_{D^k}^+(h) \subseteq A_1^h$. □

We demonstrate the impact of this arc-discretization approach in Section 3.4.

3.3.3 General SND instances

When implementing an arc-based DDD algorithm, we would like to work with the *minimally granular* arc partition, defined as follows.

Definition 3.12. *An arc partition $\mathcal{A} = \{\mathcal{A}_v\}_{v \in N}$ is* minimally granular *if subdividing any part in $\mathcal{A}$ into two parts is no longer a valid partition.*

In general, the minimally granular arc partition may be simply the partition $\mathcal{A} = \{\mathcal{A}_v\}_{v \in V}$, where $\mathcal{A}_v = \delta_D^+(v)$. However, the modified DDD approach still has potential to offer speed-up over the original implementation since each node will still have two copies in the auxiliary network – one representing the departing arcs, and the other representing the destination node. For settings where nodes serve as destinations for some commodities and intermediate nodes for others, this still allows additional freedom in the selection of departure times, reducing the size of the partial networks in the DDD algorithm.

We also note that given sets $D^k \subseteq D$ for each commodity $k \in \mathcal{K}$, the minimally granular valid partition of the arc set can be computed efficiently. The following algorithm gives

the pseudocode for this process.

Algorithm 6: `arc-partition`$(D, \{D^k\}_{k\in\mathcal{K}})$

Input: Base network $D = (N, A)$, and subgraph D^k for each commodity $k \in \mathcal{K}$

$\mathcal{A}_v \leftarrow \{\{a\} : a \in \delta^+_D(v)\}$ for all $v \in N$
for $k \in \mathcal{K}$ **do**
  for $v \in N$ **do**
    $\mathcal{A}'_v \leftarrow \{A_i \in \mathcal{A}_v : \delta^+_{D^k}(v) \cap A_i \neq \emptyset\}$
    Merge all sets in $\mathcal{A}'_v$ and update $\mathcal{A}_v$
$\mathcal{A} \leftarrow \{\mathcal{A}_v\}_{v\in N}$

Standard preprocessing approaches can be used to generate the subgraph D^k for each $k \in \mathcal{K}$. For example, all arcs in $\delta^+_D(d_k)$ can be removed from D for commodity k. Similarly, every arc vw can be removed from D for commodity k if D contains no w, d_k-dipath. In the same vein, arcs vw can be removed from D for commodity k if D contains no o_k, v-dipath.

3.4 Computational Results

In this section we compare the performance of the arc-based DDD approach to the original node-based DDD approach. The two algorithms are applied to a variety of SND-RR instances in order to gain a better understanding of the factors that impact the comparative performance. The instances considered fall into three classes: SND-RR where each commodity has a designated flat path (Section 3.4.2), SND where the flat network is a hub-and-spoke network (Section 3.4.3), and SND on random flat networks where release times and deadlines are restricted to a set of critical times (Section 3.4.4). The latter two settings are instances of SND-RR where there are no restrictions on the physical route taken by a commodity.

In Section 3.4.1 we provide an overview of the construction of the SND instances used in [2], which we use for our instances of SND with designated paths, and instances of SND with critical times. The instances on hub-and-spoke networks require a different construction of the flat network which we present in Section 3.4.3. Each algorithm was coded in Python 3.6.9 with Gurobi 8.1.1 [26] as the optimization solver. The running time limit was set to 10,800 seconds (three hours) using the deterministic option of the solver and the instances were solved on three cores to within 1% of optimality. The instances were run in a 64 cores 2.6GHz Xeon Gold 6142 Processor with 256GB RAM, running a Linux operating system.

The generated instances as well as the implementation of the arc-based and node-based DDD approaches can be found at https://github.com/madisonvandyk/SND-RR.

3.4.1 SND baseline instances

We generate the instances as constructed by Boland et al. [2]. These temporal instances were modified from the flat instances of Crainic et al. [10, 11] that are frequently used as benchmarking instances for static SND. In this construction, arcs (A) and origin-destination pairs forming commodities ($\mathcal{K}$) are chosen randomly. The capacities (u), demands (q), and fixed and variable costs (f and c) are then chosen independently at random from uniform distributions. Afterwards, these values are scaled so that the specified *cost ratio*, F, and *capacity ratio*, C, are achieved where Q is the total demand and

$$F := |\mathcal{K}| \sum_{ij \in A} \frac{f_{ij}}{Q \sum_{k \in \mathcal{K}} \sum_{ij \in A} c_{ij}^k} \quad \text{and} \quad C := \frac{|A|Q}{\sum_{ij \in A} u_{ij}}.$$

The values of these ratios considered in [10, 11] are $F \in \{0.01, 0.05, 0.1\}$ and $C \in \{1, 2, 8\}$. The parameters chosen to generate flat instances in our experiments are presented in Sections 3.4.2, 3.4.3, and 3.4.4, and differ due to the varying difficulty of each family of instances. For example, when a physical path is designated for each commodity, this significantly decreases the number of variables, and so the number of nodes, arcs, and commodities are increased in order to obtain instances that are sufficiently large. For the precise construction of the flat instances see [10].

Boland et al. [2] create timed instances by assigning transit times τ to arcs, and generating release times and deadlines for the commodities. The arc transit times are simply a scaling of the fixed costs. Given the transit times τ, they compute the average transit time of the shortest commodity path, $\mathcal{L}$. The release times are then chosen according to a normal distribution with mean $\mathcal{L}$ and standard deviation σ_r. The deadline is set to be $r_k + \mathcal{L} + p_k$, where p_k is the level of flexibility and is chosen from another normal distribution with mean μ_p and standard deviation $\frac{1}{6}\mu_p$. While various discretization levels, Δ, are studied in [2], we use only $\Delta = 1$ since our computational study compares runtime and iteration sizes rather than the degradation of the optimal value in coarser discretizations. Again, the parameters considered vary for each instance family, and so they are presented in the beginning of each subsection.

3.4.2 SND with designated paths

For each set of parameters in Table 3.1 we generate a flat instance. Then, for each of the 32 flat instances, we create 6 SND-RR instances according to the parameters in Table 3.2, for a total of 192 instances. To create instances of SND-RR where each designated subgraph D^k is a single path P_k, for each commodity $k \in \mathcal{K}$ we assign the path P_k to be the a shortest o_k, d_k-dipath by transit time. We note that while variable costs could be removed from the model when paths are fixed, higher variable costs relative to fixed costs allow the algorithms to terminate earlier, and so this comparison is still valuable.

$\|N\|$	$\|A\|$	$\|\mathcal{K}\|$	F	C
20	230, 300	150, 200	0.05, 0.1	1, 8
25	360, 480	250, 300	0.05, 0.1	1, 8

Table 3.1: Flat instance parameters.

Normal distribution	μ	σ
For generating r_k	$\mathcal{L}$	$\frac{1}{3}\mathcal{L}, \frac{1}{6}\mathcal{L}, \frac{1}{9}\mathcal{L}$
For generating p_k	$\frac{1}{4}\mathcal{L}, \frac{3}{8}\mathcal{L}$	$\frac{1}{6}\mu$

Table 3.2: Timed instance parameters.

Computational results

In Table 3.3 we present the deciles for the runtime of each algorithm, and for deciles where the algorithms did not terminate within the time limit, we instead compare the optimality gap found after terminating the algorithms after three hours. For each decile we see that the arc-based DDD approach offers a significant improvement over the traditional node-based DDD approach. The majority of the instances (182 out of 192) were solved to within 1% of optimality within three hours by both the arc-based and node-based DDD algorithms. Among these instances, the arc-based DDD approach completed in 42.6% of the time of the node-based approach, representing an improvement of 57.4% in runtime.

	ave. total runtime (s)			ave. optimality gap		
Decile	arc-based	node-based	% improvement	arc-based	node-based	% improvement
0.1	42	138	69.2%	< 1%	< 1%	N/A
0.2	75	239	68.5%	< 1%	< 1%	N/A
0.3	128	437	70.6%	< 1%	< 1%	N/A
0.4	173	629	72.4%	< 1%	< 1%	N/A
0.5	250	833	70.0%	< 1%	< 1%	N/A
0.6	451	1,188	62.0%	< 1%	< 1%	N/A
0.7	727	1,832	60.3%	< 1%	< 1%	N/A
0.8	1,252	3,328	62.4%	< 1%	< 1%	N/A
0.9	3,142	7,247	56.6%	< 1%	< 1%	N/A
1	10,800	10,800	N/A	5.87%	8.49%	30.8%

Table 3.3: Runtime and optimality gap comparison for designated path instances.

The fraction of the instances solved over time is presented in Figure 3.9, and we again observe a consistent improvement of the arc-based approach over the node-based approach. In Table 3.4 we present the iteration statistics of each algorithm considering only the 182 instances which were solved within the time limit for both algorithms. The table reports the average number of iterations until termination, and the average number of variables and constraints in the *final iteration*. The ratio for each field is the arc-based value to the node-based value. While the arc-based approach terminates after on average 18% additional iterations of DDD, the runtime decreases since the formulation solved in each iteration in the arc-based approach is smaller; on average there are only 55% the number of variables and 60% the number of constraints in the final iteration compared to the node-based DDD approach.

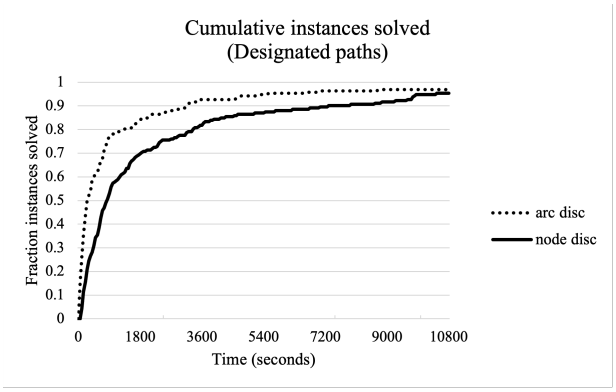


Figure 3.9: Instances solved over time.

	arc-based	node-based	ratio
# iterations	24.1	20.4	1.18
# variables	41,640	75,112	0.55
# constraints	28,375	47,567	0.60

Table 3.4: Iteration comparison.

It is worth noting that we report the number of variables and constraints in the formulation in the final iteration for each approach, rather than the number of timed nodes and timed arcs in the final partial network. The size of the partial networks are not directly comparable due to the increased number of arcs in the auxiliary network G compared to the flat network D. In contrast, the variables and constraints are comparable and have an impact on the time required to solve the corresponding formulation.

Figures 3.10 and 3.11 demonstrate the performance of the two algorithms over the iterations. Figure 3.10 shows the average number of variables in the lower bound formulation solved in each iteration for both the node-based and arc-based DDD algorithms. Similarly, Figure 3.11 shows the average percentage gap between the upper and lower bounds for each iteration. The data includes all instances for which each algorithm terminated within the time limit. In each iteration, the average size of the (lower bound) formulation solved in the arc-based DDD approach is significantly smaller than that of the node-based approach.

This difference is more significant than the difference between the average percentage gap between the lower bound and upper bound in each iteration for the two algorithms. Observe that for all percentage gaps, the size of the formulation in the iteration that achieves that gap is smaller in the case of the arc-based approach.

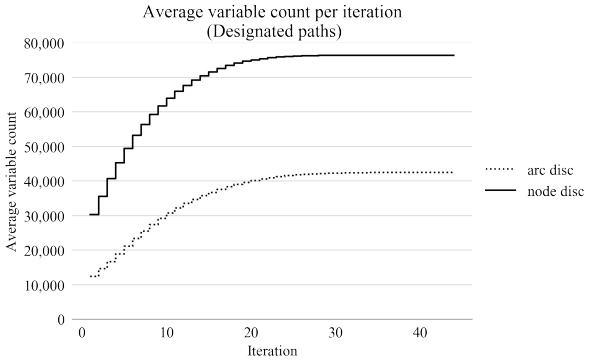


Figure 3.10: Ave. variable count per iteration.

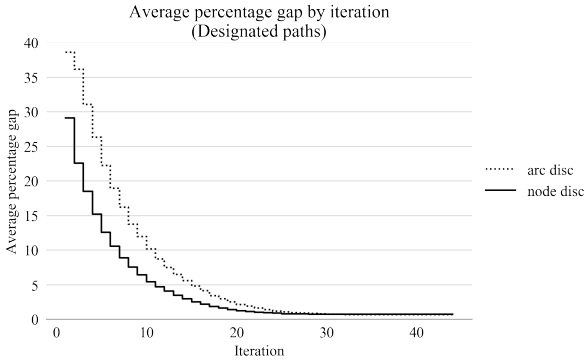


Figure 3.11: Ave. percentage gap per iteration.

Since the formulation sizes are significantly smaller, it takes less time for the arc-based approach to reach a fixed percentage gap between the lower and upper bounds. A characteristic instance is provided in Figure 3.12. Observe that the horizontal steps (the time for each iteration) are much shorter than the time for the node-based iterations.

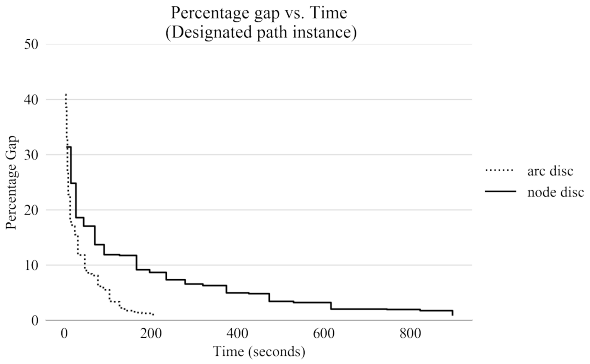


Figure 3.12: Percentage gap between upper and lower bounds over time.

3.4.3 SND on hub-and-spoke networks

In hub-and-spoke networks, regions are typically formed by selecting a hub for nearby nodes [4]. The flat networks constructed by Crainic et al. do not allow for consistent distance-based assignment of nodes to hubs to form regions since the arc distances are selected uniformly at random. For this reason, we instead form flat networks using a geometric approach. A similar method is used to study the relaxation of node storage constraints in Section 4.4.2.

In this geometric approach, n nodes are randomly chosen from an $l \times l$ grid ($l = 30$). Then from this set of nodes, h nodes are randomly selected and designated as hubs. Let $\mathcal{H}$ denote the set of hubs. Regions are formed by assigning each node to the nearest hub by L1-norm distance. We add all arcs between hub nodes and the remaining $|A| - |\mathcal{H}|(|\mathcal{H}| - 1)$ regional arcs are selected at random. We assign the transit time of the arcs in the network to be equal to the L1-norm distance between the endpoints. Note that the number of possible arcs decreases as we move to a hub-and-spoke network since all arcs are either regional or core arcs. In Section 3.4.1, the values for $|A|$ were chosen so that instances had either approximately 60% or 80% of the total possible arcs. In keeping with this density, the following arc counts are chosen so that the number of arcs chosen as either 60% or 80% of the expected total possible arcs in the hub-and-spoke network. The fixed costs, variable costs, and commodities are chosen as in Section 3.4.1 to form flat instances. For each of the 32 flat instances with parameters in Table 3.5, we create 6 instances of SND-RR with the parameters in Table 3.6 for a total of 192 instances.

$\|N\|$	$\|\mathcal{H}\|$	$\|A\|$	$\|\mathcal{K}\|$	F	C
20	3	70, 95	100	0.05, 0.1	1, 8
20	4	55, 75	100	0.05, 0.1	1, 8
20	5	50, 65	100	0.05, 0.1	1, 8
20	6	45, 60	100	0.05, 0.1	1, 8

Table 3.5: Flat instance parameters.

Normal distribution	μ	σ
For generating r_k	$\mathcal{L}$	$\frac{1}{3}\mathcal{L}, \frac{1}{6}\mathcal{L}, \frac{1}{9}\mathcal{L}$
For generating p_k	$\frac{1}{4}\mathcal{L}, \frac{3}{8}\mathcal{L}$	$\frac{1}{6}\mu$

Table 3.6: Timed instance parameters.

Computational results

For hub-and-spoke instances we also observe a significant improvement in runtime when using the arc-based DDD approach. In Table 3.7 we present the deciles for the runtime and optimality gap of each algorithm after three hours. The majority of the instances (139 out of 192) were solved to within 1% of optimality within three hours by both the arc-based and node-based DDD algorithms. Among these instances, the arc-based DDD approach

completed in 58.2% of the time of the node-based approach, representing an improvement of 41.8% in runtime.

	ave. total runtime (s)			ave. optimality gap		
Decile	arc-based	node-based	% improvement	arc-based	node-based	% improvement
0.1	29	62	54.0%	< 1%	< 1%	N/A
0.2	83	232	64.2%	< 1%	< 1%	N/A
0.3	154	404	61.8%	< 1%	< 1%	N/A
0.4	307	832	63.1%	< 1%	< 1%	N/A
0.5	463	1,514	69.4%	< 1%	< 1%	N/A
0.6	1,310	4,096	57.3%	< 1%	< 1%	N/A
0.7	4,219	7,249	41.8%	< 1%	< 1%	N/A
0.8	10,727	10,800	N/A	1.03%	11.0%	90.6%
0.9	10,800	10,800	N/A	19.4%	42.1%	53.9%
1	10,800	10,800	N/A	50.4%	62.6%	19.4%

Table 3.7: Runtime and optimality gap comparison for hub-and-spoke instances.

The fraction of the instances solved over time is presented in Figure 3.9, and we again observe a consistent improvement of the arc-based approach over the node-based approach. In Table 3.4 we present the iteration statistics of each algorithm considering only the 139 instances which were solved within the time limit for both algorithms. The arc-based approach terminates after on average 2% more iterations, and on average there are only 66% the number of variables and 69% the number of constraints in the final iteration compared to the node-based DDD approach.

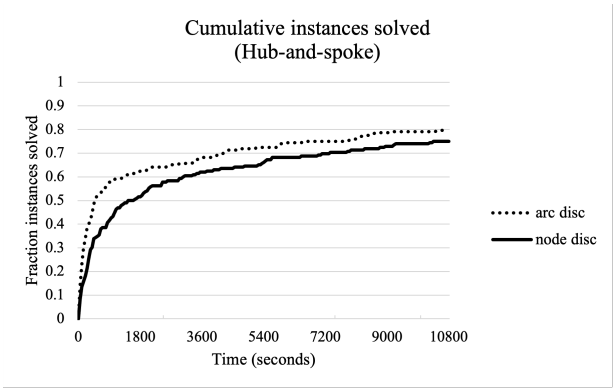


Figure 3.13: Fraction of instances solved.

	arc-based	node-based	ratio
# iterations	23.4	22.9	1.02
# variables	17,102	26,088	0.66
# constraints	8,835	12,715	0.69

Table 3.8: Iteration comparison.

The comparative performance is consistent with the performance on the designated path instances. Figures 3.14 and 3.15 show the average number of variables in the lower bound

formulation, and average percentage gap between the upper and lower bounds, respectively, for each iteration. The data includes all instances for which each algorithm terminated within the time limit.

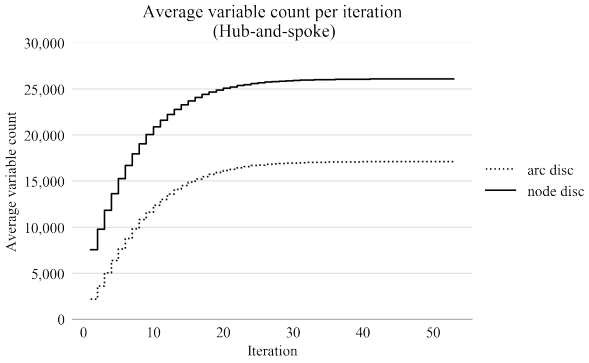


Figure 3.14: Ave. variable count per iteration.

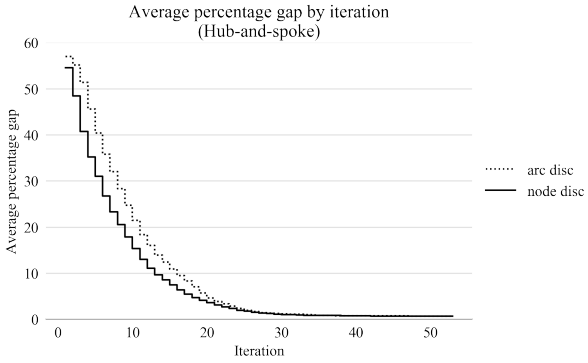


Figure 3.15: Ave. percentage gap per iteration.

We see that in general it takes less time for the arc-based approach to reach a fixed percentage gap between the lower and upper bounds. A characteristic instance is provided in Figure 3.16.

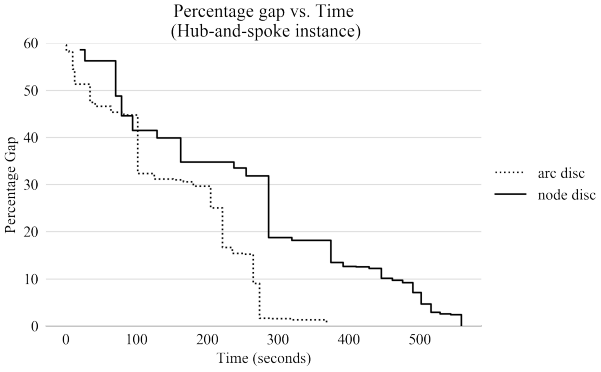


Figure 3.16: Percentage gap between upper and lower bounds over time.

3.4.4 SND with critical times

In many practical LTL instances, freight is available in batches due to employee shift rotations. For instance, freight may be released at a fixed warehouse at 7pm, and due by 9pm at another warehouse. The instances in this section reflect the reality that many

release times and deadlines fall into a subset of *critical* times. To create instances with critical times, we introduce a parameter, α, which is the number of intervals the time horizon is split into evenly. Then, for each node, a critical time is chosen uniformly at random from each of the intervals. We then modify the release times and deadlines of each commodity so that release times are rounded down to the closest critical time at that node, and deadlines are rounded up to the nearest critical time at that node. This construction allows us to capture node-dependent critical times which add structure to the SND instances. In total, we create 16 flat instances according to Table 3.9 and for each flat instance we create 12 SND instances according to the parameters in Table 3.10 and $\alpha \in \{5, 10\}$.

$\|N\|$	$\|A\|$	$\|\mathcal{K}\|$	F	C
20	230, 300	150, 200	0.05, 0.1	1, 8

Table 3.9: Flat instance parameters.

Normal distribution	μ	σ
For generating r_k	$\mathcal{L}$	$\frac{1}{3}\mathcal{L}, \frac{1}{6}\mathcal{L}, \frac{1}{9}\mathcal{L}$
For generating p_k	$\frac{1}{4}\mathcal{L}, \frac{3}{8}\mathcal{L}$	$\frac{1}{6}\mu$

Table 3.10: Timed instance parameters.

Computational results

For instances with critical times we observe a moderate improvement in runtime when using the arc-based DDD approach. In Table 3.11 we present the deciles for the runtime and optimality gap of each algorithm after three hours. 174 out of 192 instances were solved to within 1% of optimality within three hours by both the arc-based and node-based DDD algorithms. Among these instances, the arc-based DDD approach completed in 87.6% of the time of the node-based approach, representing an improvement of 12.4% in runtime. While the gap in the final row is larger in the arc-based approach, this decile only captures the maximum value. Since all the previous deciles consistently show that the arc-based approach improves over the node-based approach, we do not find this entry a convincing argument against the arc-based approach.

	ave. total runtime (s)			ave. optimality gap		
Decile	arc-based	node-based	% improvement	arc-based	node-based	% improvement
0.1	72	91	20.6%	< 1%	< 1%	N/A
0.2	117	153	23.6%	< 1%	< 1%	N/A
0.3	192	247	20.0%	< 1%	< 1%	N/A
0.4	312	375	16.7%	< 1%	< 1%	N/A
0.5	420	485	13.4%	< 1%	< 1%	N/A
0.6	701	867	19.2%	< 1%	< 1%	N/A
0.7	1,131	1,256	10.0%	< 1%	< 1%	N/A
0.8	2,330	2,775	16.0%	< 1%	< 1%	N/A
0.9	7,328	7,469	1.9%	< 1%	< 1%	N/A
1	10,800	10,800	N/A	44.6%	36.3%	-22.9%

Table 3.11: Runtime and optimality gap comparison for instances with critical times.

In Table 3.12 we present the iteration statistics of each algorithm considering only the 174 instances which were solved within the time limit for both algorithms. The arc-based approach terminates after on average 17% more iterations, and on average there are only 80% the number of variables and 81% the number of constraints in the final iteration compared to the node-based DDD approach.

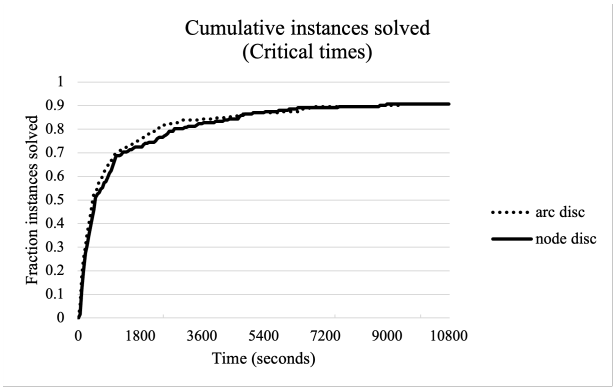


Figure 3.17: Fraction of instances solved.

	arc-based	node-based	ratio
# iterations	18.4	15.7	1.17
# variables	34,794	43,697	0.80
# constraints	17,696	21,934	0.81

Table 3.12: Iteration comparison.

Figures 3.18 and 3.19 show the average number of variables in the lower bound formulation, and average percentage gap between the upper and lower bounds, respectively, for each iteration. The data includes all instances for which each algorithm terminated within the time limit. The comparative performance is consistent with the performance on the designated path instances, albeit less pronounced. This is not surprising, since when the arc partition is not particularly fine, the arc-based and node-based approaches are more similar.

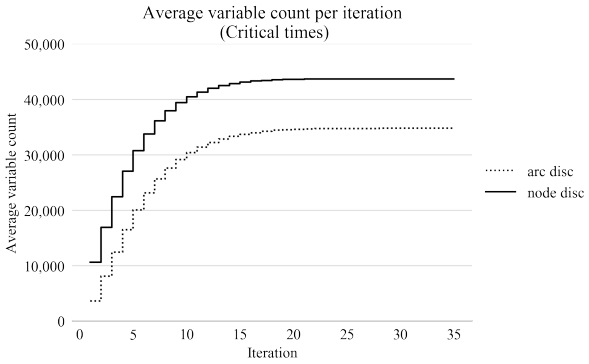


Figure 3.18: Ave. variable count per iteration.

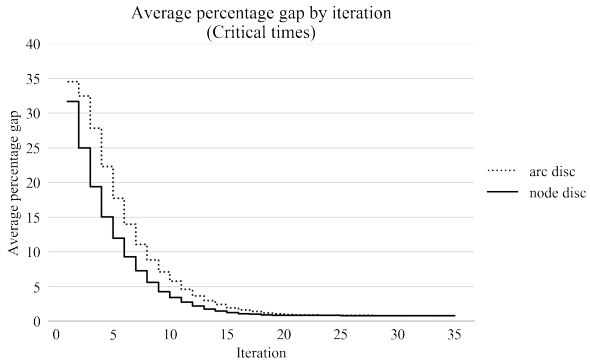


Figure 3.19: Ave. percentage gap per iteration.

A characteristic instance is provided in Figure 3.20. Observe that the arc-based DDD approach still outperforms the node-based approach for the majority of the gap targets, but not all.

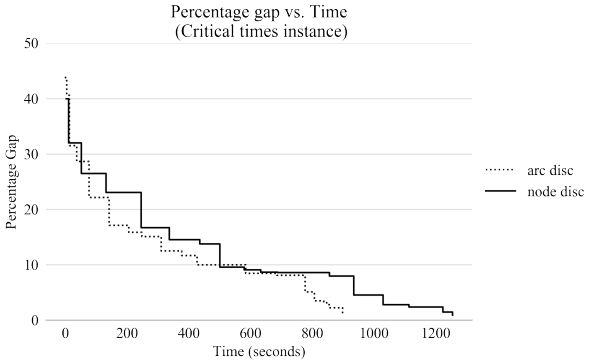


Figure 3.20: Percentage gap between upper and lower bounds over time.

3.5 Summary and observations

Prior to our work, DDD was ineffective for solving problems where all near-optimal solutions require a large number of departure times at many nodes in the network. In this work, we present a novel algorithmic approach for generating smaller lower bound formulations in each iteration of DDD. This reduction in iteration size is accomplished by allowing an arc-dependent set of departure times, rather than a fixed set of departure times for each node. Moreover, the approach presented builds upon the original DDD paradigm so that

the modifications required to build the arc-based DDD algorithm from a node-based DDD algorithm are minimal. The experimental results highlight the effectiveness of this DDD approach, as well as introduce important design considerations for future benchmarking instances such as regional networks and grouping release times and deadlines using critical times at nodes, which model real-world instances more closely.

The arc-based DDD approach presented in this chapter offers the most improvement over the original node-based DDD approach when the arc partitioning is fine. However, for certain problems with unrestricted routes, such a fine partition of the arc set is unlikely. The discussion in the opening of Section 3.1 points to trade-offs in allowing independent departure times for arcs in $\delta^+(v)$ in the flat network. Namely, without allowing independence of arc departure times through a valid arc partition, this splitting would result in additional timed copies of arcs in $\delta^-(v)$. When $\delta^-(v) = \emptyset$, there is no downside and the definition of a valid arc partition can be relaxed for these nodes. It is possible that when the in-degree at a node v is smaller than the out-degree, allowing arc-dependent departure times is advantageous even when it results in commodities with variables for multiple copies of the same arc. Moreover, it is possible that an arc partition that changes over each iteration of DDD improves the performance of the algorithm. Characterizing when these design choices should be made is an interesting research direction.

Finally, there are many problems for which all near-optimal solutions require a large number of departure times for the majority of the arcs. That is, solutions can only be expressed on large subgraphs of the full time-expanded network. For such problems, even the arc-based DDD approach presented here is unlikely to be find an optimal solution more quickly than solving the full time-indexed formulation. However, many temporal problems exhibit cyclic and repetitive behaviour. An interesting direction of research would be to incorporate repetitive flows into the DDD framework, without leading to an increased formulation size.

Chapter 4

Node storage

In this chapter we present a DDD algorithm for a problem with static node storage constraints. We present efficiently computable bounds on the required storage increase at each timed node, given the input instance and the current partial network. Moreover we establish these bounds in a problem-agnostic way, ensuring that they are broadly applicable to problems with static node storage. We also demonstrate the importance of finding tight bounds; simpler analytical arguments may yield bounds, but these weaker bounds can cause a DDD algorithm to require many more iterations until terminating.

4.1 Problem statement and background

In this section we introduce the problem of universal packet routing along with a time-indexed formulation for the problem. Note that for this and remaining chapters, we work with the generic node-based DDD approach.

4.1.1 Universal packet routing

Let $D = (N, A)$ be a directed graph which we will call the *flat* or *base* network. Each arc $vw \in A$ has an associated transit time $\tau_{vw} \in \mathbb{N}$, and a capacity $u_{vw} \in \mathbb{N}$ which denotes the maximum number of packets that can depart along arc vw simultaneously. Let $\mathcal{K}$ denote a set of packets and for each packet $k \in \mathcal{K}$, let o_k and d_k denote its associated origin and destination respectively. We say that a packet is *active* if it is not located at its origin or destination. Additionally, each node $v \in N$ has a storage level of $b_v \in \mathbb{N}$, meaning that it can store at most b_v active packets at any time. The makespan of a schedule is

the latest arrival time of any packet at its destination. The objective of *universal packet routing* (UPR) is to send each packet along a single trajectory in D_∞, the infinite fully time-expanded network where $T = \infty$, that minimizes the makespan of the schedule while respecting arc capacity and node storage. We let T^* denote the minimum makespan. Note, the abstract notion of D_∞ is only used to define the packet routing problem.

We note that the techniques presented in this chapter extend to the setting where packet sizes are also of arbitrary size, as well as the setting where packets have varying release times. In this chapter, all packets become available at the start of the time horizon.

Full time-indexed formulation

In order to map an instance of UPR to a finite time-expanded network, we need to know some upper bound T on the value of the minimum makespan, T^*. Unfortunately, UPR is at least as hard to approximate as vertex colouring [5]. Thus, in practical applications we are forced to use a relatively large value of T as an upper bound to ensure a solution can be found in the corresponding fully time-expanded network D_T. We show in Section 4.4 that this can greatly increase solving time, partly due to the introduction of additional symmetries in the corresponding MIP as T increases. When solving an instance of UPR that arises from a practical application, we may have solved similar instances in the past and as a result know a good upper bound T on T^*, where $T = (1 + \alpha)T^*$ for a small value of $\alpha \geq 0$. In contrast, if the instance is unknown to us, the upper bound we can produce would likely require a large value for α. In our computational experiments, we therefore test the proposed algorithm with upper bounds of $T^*, 1.5T^*$, and $2T^*$ to understand the performance of DDD compared to the traditional MIP as the strength of the known upper bound varies.

Recall that for each node $v \in N$, only commodities whose origin and destination are not equal to v contribute towards the storage level at v. That is, only active commodities at v contribute to storage. For each $v \in N$, let $\mathcal{K}_v$ denote the set of commodities that are *active* at v. That is, $\mathcal{K}_v = \{k \in \mathcal{K} : o_k \neq v, d_k \neq v\}$. Note that the commodities in $\mathcal{K} \setminus \mathcal{K}_v$ do not contribute to storage levels at node v, since they are either at their origin or at their destination. We emphasize that $\mathcal{K}_v$ is *not* time-dependent.

Let T be an upper bound on the value of T^*, which we will assume is given to us. We would like to determine trajectories in D_T that minimizes makespan while obeying the capacity limitations. First, note that since all input data is integer, the decision times of an optimal solution are in $[T]$. For each packet $k \in \mathcal{K}$ and each timed arc $e \in A_T \cup H_T$, we have a binary variable x_e^k which is equal to 1 if packet k is scheduled to travel along timed arc e

in its assigned trajectory in D_T. We assign the timed arc capacities in D_T directly from the arc and node capacities in D. Specifically, for each timed arc $e = ((v,t),(w,t')) \in A_T$, we define $u_e := u_{vw}$. Similarly, for $e = ((v,t)(v,t')) \in H_T$, we define $b_e := b_v$.

Let $\delta^+_{D_T}(v,t)$ and $\delta^-_{D_T}(v,t)$ denote the outgoing and incoming timed arcs at (v,t) in D_T. That is, $\delta^+_{D_T}(v,t) = \{e \in A_T \cup H_T : e = ((v,t),(w,t'))\}$, and $\delta^-_{D_T}(v,t) = \{e \in A_T \cup H_T : e = ((w,t'),(v,t))\}$. The following IP models UPR when we are given the upper bound T.

$$\min \quad \bar{T} \tag{UPR(D_T)}$$

$$\text{s.t.} \quad t' \cdot x^k_e \le \bar{T} \quad \forall k \in \mathcal{K},\ \forall e = ((v,t),(w,t')) \in A_T \tag{4.1}$$

$$\sum_{e \in \delta^+_{D_T}(v,t)} x^k_e - \sum_{e \in \delta^-_{D_T}(v,t)} x^k_e = \begin{cases} 1 & (v,t) = (o_k, 0) \\ -1 & (v,t) = (d_k, T) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_T \tag{4.2}$$

$$\sum_{k \in \mathcal{K}} x^k_e \le u_e \quad \forall e \in A_T \tag{4.3}$$

$$\sum_{k \in \mathcal{K}_v} x^k_e \le b_e \quad \forall\ e \in H_T \tag{4.4}$$

$$x^k_e \in \{0,1\} \quad \forall k \in \mathcal{K},\ \forall e \in A_T \cup H_T. \tag{4.5}$$

Constraint (4.1) encodes that the time horizon is equal to the latest arrival time among all packets. Constraint (4.2) ensures that the set of trajectories satisfies flow conservation. Finally, constraints (4.3) and (4.4) ensure that the trajectories satisfy arc capacities and node storage constraints respectively.

In addition, we will add the following constraint to strengthen the linear relaxation. This turns out to be essential when applying the two-phase DDD approach presented in Section 4.6. For each $k \in \mathcal{K}$, let $A^{final}_{T,k} = \{((v,t),(w,t')) \in A_T : w = d_k\}$ be the set of timed movement arcs entering the destination of packet k. Since each packet has a single trajectory in a feasible integer solution, we have the following constraint.

$$\sum_{e=((v,t),(w,t')) \in A^{final}_{T,k}} t' \cdot x^k_e \le \bar{T} \quad \forall k \in \mathcal{K}. \tag{4.6}$$

In the application of DDD to solve TSP with time windows [62], the authors also consider a makespan objective, encoded analogously to constraint (4.6) for a single commodity. The remainder of our lower bound, upper bound, and refinement are not comparable due to the addition of the arc and node capacity constraints in UPR.

4.1.2 DDD for universal packet routing

A sketch of the overall DDD algorithm for universal packet routing is as follows. We present the precise algorithm at the end of Section 4.3.2.

Algorithm 7: UPR-DDD$(D, \mathcal{K}, T)$

Input: Base network $D = (N, A)$, packet set $\mathcal{K}$, an upper bound, T, on the optimal makespan

Initialization: $D_S \leftarrow D_0$, where UPR(D_S) provides a lower bound for UPR(D_T)
while *not solved* **do**
  Lower bound: Solve UPR(D_S) and obtain a solution $\hat{x}$ in D_S
  Upper bound/termination: determine if $\hat{x}$ can be converted to a solution to UPR(D_T) with the same makespan
  if *yes* **then**
    Stop. An optimal solution has been found for UPR(D_T).
  Refinement: update D_S while ensuring UPR(D_S) provides a lower bound for UPR(D_T).

As described above, the key components of the DDD approach are the lower bound model, the upper bound/termination step, and the refinement step. In the lower bound model, partial networks are constructed with properties (P1) and (P2) (see Section 2.1) so that arc transit times are *underestimated* and every trajectory in D_T can be mapped to a trajectory in D_S with the same underlying path, albeit with shortened timed arcs. In Section 4.2 we present two additional properties on the relaxation of arc and node storage capacities required to ensure UPR(D_S) is a relaxation of UPR(D_T). In Section 4.3, we describe the upper bound model and augmentation steps. Computational results are presented in Section 4.4.

4.2 Lower Bound Model

One of the key components of the DDD iterative approach is the lower bound model. Specifically, given an appropriate subset of timed nodes N_S, we want to obtain a partial network $D_S = (N_S, A_S \cup H_S)$ along with a formulation UPR(D_S) which has optimal value at most that of UPR(D_T). As is standard in DDD, we construct A_S according to (P1) and (P2). For universal packet routing, we will assume we are given an upper bound, T, on the minimum required makespan, and we set $r_k = 0$ and $l_k = T$ for all $k \in \mathcal{K}$.

(P1) **Timed nodes**: For all $k \in \mathcal{K}$, (o_k, r_k) and (d_k, l_k) are in N_S;
For all $v \in N$, $(v, 0)$, and (v, T) are in N_S;
(P2) **Arc copies**: For all $vw \in A$, for all $(v,t) \in N_S$ with $t + \tau_{vw} \leq T$, we have
$((v,t),(w,t')) \in A_S$ where $t' = \max\{r : r \leq t + \tau_{vw}, (w,r) \in N_S\}$.

Note that these properties alone do not ensure that an optimal routing of the packets through D_S provides a lower bound on the optimal routing through D_T, due to the arc and node capacity constraints. For example, if the timed arcs in D_S were all given the same capacity as their underlying arc in D, then D_S would have a smaller total arc capacity than D_T. We now state the lower bound formulation corresponding to a partially time-expanded network D_S, and define u' and b' in Sections 4.2.1 and 4.2.2. The techniques presented in this paper can easily be extended to the setting where r_k and l_k vary, as well as the case where commodities have non-unit demands and flow for a single commodity can be sent fractionally along multiple trajectories. In this chapter, $r_k = 0$ for each commodity $k \in \mathcal{K}$.

$$\min \quad \bar{T} \tag{UPR(D_S)}$$

$$\text{s.t.} \quad (t + \tau_{vw}) \cdot x_e^k \leq \bar{T} \quad \forall k \in \mathcal{K},\ \forall e = ((v,t),(w,t')) \in A_S \tag{4.7}$$

$$\sum_{e=((v,t),(w,t')) \in A_{S,k}^{final}} (t + \tau_{vw}) \cdot x_e^k \leq \bar{T} \quad \forall k \in \mathcal{K}. \tag{4.8}$$

$$\sum_{e \in \delta_{D_S}^+(v,t)} x_e^k - \sum_{e \in \delta_{D_S}^-(v,t)} x_e^k = \begin{cases} 1 & (v,t) = (s_k, 0) \\ -1 & (v,t) = (t_k, T) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_S \tag{4.9}$$

$$\sum_{k \in \mathcal{K}} x_e^k \leq u'_e \quad \forall e \in A_S \tag{4.10}$$

$$\sum_{k \in \mathcal{K}_v} x_e^k \leq b'_e \quad \forall e \in H_S \tag{4.11}$$

$$x_e^k \in \{0,1\} \quad \forall k \in \mathcal{K}, e \in A_S \cup H_S. \tag{4.12}$$

In addition to modifying the arc and node capacities, we replaced constraints (4.1) and (4.6) in UPR(D_T) with (4.7) and (4.8). Observe that in a partially time-expanded network, we may have $t' < t + \tau_{vw}$ for some $((v,t),(w,t')) \in A_S$. Therefore, constraints (4.7) and (4.8) are tighter than constraints (4.1) and (4.6). In Section 4.3.2, we will prove that constraint (4.7) ensures that throughout DDD, so long as $\tau_a \geq 1$ for all $a \in A$, we never add a timed node (v,t) with $t > T^*$.

We need to assign the arc capacities and node storage levels in D_S to ensure that UPR(D_S) is a relaxation of UPR(D_T). As is standard, to prove that the optimal value of UPR(D_S) is

at most that of UPR(D_T), we use the map μ (equation 2.10), as defined in Section 2.2, to map feasible solutions of UPR(D_T) with makespan $\bar{T}$ to those of UPR(D_S) with makespan at most $\bar{T}$.

Recall from (2.10), for any timed arc $e = ((v,t),(w,t')) \in A_T$, $\mu(e) = ((v,\hat{t}),(w,\hat{t}'))$ where $\hat{t} = \max\{s : s \leq t, (v,s) \in N_S\}$ and $\hat{t}' = \max\{s : s \leq \hat{t} + \tau_{vw}, (v,s) \in N_S\}$. With this map μ in mind, we will show how to define u', and b' so that UPR(D_S) is a relaxation of UPR(D_T). Let $\bar{x}$ be a feasible solution to UPR(D_T). We define $\hat{x}$ as the binary vector such that for all $e \in A_S$ and $k \in \mathcal{K}$,

$$\hat{x}_e^k = \max\{\bar{x}_f^k : \mu(f) = e\}.$$

That is, for all $e \in A_S$ and $k \in \mathcal{K}$, $\hat{x}_e^k = 1$ if $\mu(f) = e$ for any timed arc $f \in A_T$ with $\bar{x}_f^k = 1$. By abuse of notation, $\hat{x}$ obtained from $\bar{x}$ in this manner is denoted by $\mu(\bar{x})$ in this paper.

In the seminal work of Boland et al. first introducing the DDD method, the authors prove the following lemma for problems with flow conservation constraints (Theorem 2 in [2]). This proof is shown in the proof of Theorem 2.2 in Chapter 2.

Lemma 4.1. *If $\bar{x}$ is a vector that satisfies the flow and integrality constraints in UPR(D_T) (constraints (4.2) and (4.5)), then $\hat{x} = \mu(\bar{x})$ satisfies the analogous constraints in UPR(D_S) ((4.9) and (4.12)).*

4.2.1 Arc capacities

Techniques to incorporate arc capacities into the DDD framework were presented in [39] and [50]. We present our work in full in this section for clarity, and for use in the novel work in Section 4.2.2.

In the example provided in Figure 4.1, we assume there are unit arc capacities. Observe that μ will map each of the two $u \to v \to w$ trajectories (represented with dashed blue lines) to the same trajectory in D_S, which exceeds the original unit capacities. Thus, it is necessary to add to properties (P1) and (P2) in order to provide a lower bound for problems with arc capacities.

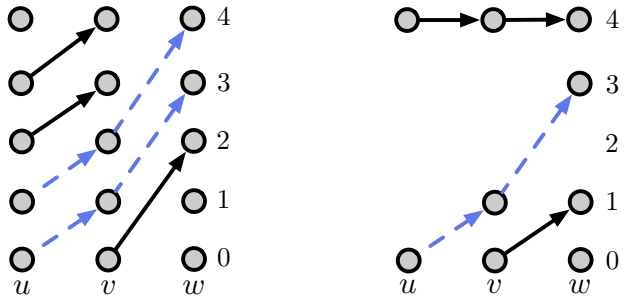


Figure 4.1: Two trajectories in D_T are mapped to the same trajectory in D_S via μ.

For each timed node $(v,t) \in N_S$, let $\mathtt{n_S}(v,t)$ be the time of the first appearance of v after t in N_S. That is,

$$\mathtt{n_S}(v,t) = \min\{t' : t' > t, (v,t') \in N_S\}.$$

Let $e = ((v,t),(w,t')) \in A_S$. Then μ maps $f = ((v,t_1),(w,t_2)) \in A_T$ to e when $t_1 \in \{t, t+1, \ldots, \mathtt{n_S}(v,t) - 1\}$. To capture the length of this interval, we define

$$\mathtt{m_S}(v,t) := \mathtt{n_S}(v,t) - t,$$

which is the number of time units until the next appearance of v in N_S. Observe that for any $(v,t) \in N_T$, $\mathtt{n_T}(v,t) = t+1$ and $\mathtt{m_T}(v,t) = 1$. Note, we will later use the fact that $\mathtt{m_S}(v,t)$ is well-defined even if $(v,t) \notin N_S$. For any timed arc $((v,t),(w,t')) \in A_S$ there are $\mathtt{m_S}(v,t)$ timed arcs in A_T that are mapped to $((v,t),(w,t'))$ according to the map μ. This proves that the following property, $(P^{\mathtt{arcs}})$, is sufficient in order for $\hat{x}$ to satisfy the arc capacity constraints of UPR(D_S).

$$(P^{\mathtt{arcs}}) \text{ For any arc } e = ((v,t),(w,t')) \in A_S,\ u'_e = u_e \cdot \mathtt{m_S}(v,t)$$

Specifically, we have proven the following lemma. We include a brief formal proof for completeness.

Lemma 4.2. *Let D_S be a partial network that satisfies properties $(P1)$, $(P2)$ and $(P^{\mathtt{arcs}})$, and let $\bar{x}$ be a solution to UPR(D_T). Then $\hat{x} = \mu(\bar{x})$ satisfies constraint (4.10).*

Proof. Let $e = ((v,t),(w,t')) \in A_S$, and consider $\sum_{k \in \mathcal{K}} \hat{x}^k_e$. If $\hat{x}^k_e = 1$ for some commodity $k \in \mathcal{K}$, then $\bar{x}^k_a = 1$ for a timed arc $a = ((v,\bar{t}),(w,\bar{t}+\tau_{vw})) \in A_T$ with $\mu(a) = e$. By definition of the map μ, we know that $\bar{t} \in \{t, t+1, \cdots, \mathtt{n_S}(v,t) - 1\}$. Thus, the set

$\{a \in A_T : \mu(a) = e\}$ has size $\mathtt{m_S}(v,t)$. Therefore,

$$\sum_{k\in\mathcal{K}} \hat{x}_e^k \leq \sum_{\substack{a\in A_T:\\ \mu(a)=e}} \sum_{k\in\mathcal{K}} \bar{x}_a^k \leq u_a \cdot \mathtt{m_S}(v,t),$$

where the final inequality holds since $\bar{x}$ was a feasible solution for UPR(D_T). The result follows since $u_e = u_a$. □

4.2.2 Storage limits

First, we describe the difference between the storage constraints in universal packet routing and the storage constraints dealt with in the work of Lagos et al. [39] when solving the continuous time inventory routing problem (CIR). In the CIR problem, a company manages the inventory of its clients, and delivers product from a single facility with a set of at most m vehicles. Lagos et al. add the restriction that only a single vehicle can be at a given client location at any point in time. In essence, this is a hard storage capacity at the parking lot for the client. They prove that in each iteration, the yard capacity must be increased to be at most m. In our setting we are bounding the number of packets instead of vehicles (and in SND this would be the number of packages). As a result, flow balance constraints already imply the limit of m at each node in UPR and SND, making such an upper bound redundant. While Lagos et al. are able to prove stronger bounds for the problem of CIR, these results do not extend to UPR. For example, Proposition 13 in [39] states that if all timed arcs incoming to a timed node (v,t) have the correct length, then no additional storage is required at (v,t). However, this is not true for UPR, nor for SND, and does not follow by use of the map μ alone.

Similar to the case of arc capacities, we cannot simply assign the node capacities from the base graph to the nodes in N_S. In this section, we will show how to assign holdover arc storage b'_e to each timed arc $e \in H_S$ to ensure that UPR(D_S) is a relaxation of UPR(D_T). We first identify settings in which the original storage constraint at a timed node (v,t) must be relaxed. To obtain a relaxation, for each such timed node (v,t) we could completely remove the storage constraint. However, this will result in weak relaxations and increase the number of iterations required until termination for DDD. We then proceed to tighten this bound in Lemmas 4.3 - 4.7. In Section 4.5, we show that without this tightening, there are instances where DDD takes $\Omega(T)$ iterations, whereas with our tighter relaxation DDD terminates in a single iteration.

Recall the definition of $\mathtt{m_S}(v,t)$, which is the number of time units until the first appearance of v after time t in N_S. Note, $\mathtt{m_S}(v,t)$ is well-defined even if $(v,t) \notin N_S$. For the following

discussion, we look at the neighbours of a timed node in D_S. For $(v,t) \in N_S$, let $N_S^-(v,t)$ be the incoming neighbours of (v,t) in D_S. Specifically,

$$N_S^-(v,t) = \{(w,t') : \exists\ e = ((w,t'),(v,t)) \in A_S\}.$$

Let $\bar{x}$ be a feasible solution to UPR(D_T), and let $\hat{x}$ be the vector we obtain via the map $\mu : A_T \to A_S$ as stated at the beginning of this section. We would like to understand how the map μ could impact the storage required at some timed node $(v,t) \in N_S$.

For all $k \in \mathcal{K}$, let Q_k be the trajectory in D_T that packet k travels along according to $\bar{x}$. Q_k consists of an ordered set of movement timed arcs, $\{e_1^k, e_2^k, \ldots, e_{l_k}^k\}$, along with additional holdover arcs. Let P_k be the corresponding path in the underlying graph D. We define $\bar{t}_u^{k,out}$ and $\bar{t}_v^{k,in}$ so that $((u,\bar{t}_u^{k,out}),(v,\bar{t}_v^{k,in})) = e_j^k$, and we define $\hat{t}_u^{k,out}$ and $\hat{t}_v^{k,in}$ analogously. That is,

$$e = ((u,\bar{t}_u^{k,out}),(v,\bar{t}_v^{k,in})) \quad \to \quad \mu(e) = ((u,\hat{t}_u^{k,out}),(v,\hat{t}_v^{k,in}))$$

Consider two consecutive movement timed arcs in D_T, $e_{uv} = ((u,\bar{t}_u^{k,out}),(v,\bar{t}_v^{k,in}))$ and $e_{vw} = ((v,\bar{t}_v^{k,out}),(w,\bar{t}_w^{k,in}))$ along the trajectory Q_k. The flow on e_{uv} and e_{vw} is mapped to $\mu(e_{uv}) = ((u,\hat{t}_u^{k,out}),(v,\hat{t}_v^{k,in}))$ and $\mu(e_{vw}) = ((v,\hat{t}_v^{k,out}),(w,\hat{t}_w^{k,in}))$ respectively.

Let (v,t) be a timed copy of v in N_S. The following straightforward facts will be used to understand how the map μ impacts the storage of packet k at (v,t).

(F1) $\hat{t}_l^{k,out} = \max\{t : t \le \bar{t}_l^{k,out}, (l,t) \in N_S\}$ for $l \in \{u,v\}$;
(F2) $\hat{t}_v^{k,in} = \max\{t : t \le \hat{t}_u^{k,out} + \tau_{uv}, (v,t) \in N_S\}$;
(F3) $\hat{t}_v^{k,in} \le \bar{t}_v^{k,in}$ and $\hat{t}_v^{k,out} \le \bar{t}_v^{k,out}$.

(F1) follows by definition of μ (equation (2.10)), and (F2) follows from (F1) along with the fact that there is a uv timed arc departing $(u,\hat{t}_u^{k,out})$ in A_S that is as long as possible (property (P2)). (F3) follows from (F1) and (F2), along with the fact that $\bar{t}_v^{k,in} = \bar{t}_u^{k,out} + \tau_{uv}$.

If packet k was previously stored at (v,t) according to $\bar{x}$ ($\bar{t}_v^{k,in} \le t < \bar{t}_v^{k,out}$), then μ cannot introduce *additional* storage of packet k. So suppose packet k is not stored at (v,t) in $\bar{x}$. If $\bar{t}_v^{k,in} \le t$, then since the packet is not stored at (v,t), we also have that $\bar{t}_v^{k,out} \le t$. In this case, by fact (F3) it follows that $\hat{t}_v^{k,out} \le \bar{t}_v^{k,out} \le t$, and so packet k is not stored at (v,t) in $\hat{x}$. However, the storage of packet k could increase at (v,t) if $\bar{t}_v^{k,in} > t$, and packet k arrives at v earlier according to $\hat{x}$ than it is scheduled to arrive according to $\bar{x}$. That is, $\hat{t}_v^{k,in} < \bar{t}_v^{k,in}$. This can happen if either:

1. Flow departs u at the same time ($\hat{t}_u^{k,out} = \bar{t}_u^{k,out}$), but $(v,\bar{t}_v^{k,in}) \notin N_S$. For example, in Figure 4.3 shows a partially time-expanded network where $(u,1) \in N_S$, but $(v,2) \notin N_S$, so $\mu((u,1),(v,2)) = ((u,1),(v,1))$;

2. Flow is forced to depart early from the preceding node ($\hat{t}_u^{k,out} < \bar{t}_u^{k,out}$). For example, in Figure 4.4 shows a partially time-expanded network where $(u,1) \notin N_S$, so flow must depart u early and $\mu((u,1),(v,2)) = ((u,0),(v,1))$.

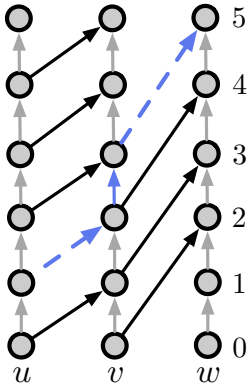


Figure 4.2: Trajectory

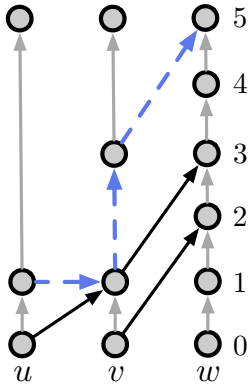


Figure 4.3: Scenario 1

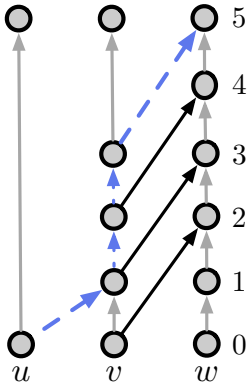


Figure 4.4: Scenario 2

Let $\mathcal{K}(v)$ denote the set of packets that travel along a trajectory that includes a timed copy of v according to $\bar{x}$. Let $\mathcal{K}^1_{\hat{x}}(v)$ denote the set of packets in $\mathcal{K}(v)$ that depart the node preceding v at the same time in $\hat{x}$ and $\bar{x}$. Similarly, let $\mathcal{K}^2_{\hat{x}}(v)$ denote the set of packets in $\mathcal{K}(v)$ that depart the preceding node earlier in $\hat{x}$ than in $\bar{x}$. Note, $\mathcal{K}(v) = \mathcal{K}^1_{\hat{x}}(v) \cup \mathcal{K}^2_{\hat{x}}(v)$. Let $e = ((v,t),(v,t'))$ be the timed arc in H_S departing (v,t), and let $f = ((v,t),(v,t+1))$ be the timed arc in H_T departing (v,t).

Lemma 4.3. *If D_S satisfies properties $(P1)$ and $(P2)$, then any packet $k \in \mathcal{K}^1_{\hat{x}}(v)$ that was not stored at v in $\bar{x}$ is not stored at v in $\hat{x} = \mu(\bar{x})$.*

Proof. Suppose packet $k \in \mathcal{K}^1_{\hat{x}}(v)$ is not stored at v in $\bar{x}$. That is, $\bar{t}_v^{k,in} = \bar{t}_v^{k,out}$. Let u be the preceding node along the path P_k in $\bar{x}$. Since $k \in \mathcal{K}^1_{\hat{x}}(v)$, we know that $\hat{t}_u^{k,out} = \bar{t}_u^{k,out}$. Thus, in D_S, packet k travels along some arc $((u,\bar{t}_u^{k,out}),(v,\hat{t}_v^{k,in}))$. By fact (F2),

$$\hat{t}_v^{k,in} = \max\{t : t \le \hat{t}_u^{k,out} + \tau_{uv}, (v,t) \in N_S\}.$$

Similarly by fact (F1),

$$\hat{t}_v^{k,out} = \max\{t : t \le \bar{t}_v^{k,out}, (v,t) \in N_S\}.$$

Since $\bar{t}_v^{k,out} = \bar{t}_u^{k,out} + \tau_{uv}$ and $\hat{t}_u^{k,out} = \bar{t}_u^{k,out}$ we see that $\hat{t}_v^{k,in} = \hat{t}_v^{k,out}$. Thus, no additional storage of packet k was introduced at v and $\hat{x}_e^k = \bar{x}_f^k = 0$. □

We first establish a simple bound on the additional storage needed at (v,t) to accommodate packets in $\mathcal{K}^1_{\hat{x}}(v)$ in Lemma 4.4. We then tighten this argument in Lemma 4.5. In Section 4.5 we show that this tightening can prove essential to the effectiveness of DDD for certain problem instances.

Lemma 4.4. *If D_S satisfies properties $(P1)$ and $(P2)$, then*

$$\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \leq \sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \bar{x}^k_f + (\mathtt{m_S}(v,t)-1)b_v.$$

Proof. Suppose packet k was stored at v in $\bar{x}$ ($\bar{t}^{k,in}_v < \bar{t}^{k,out}_v$), but not at time t. We know that as in Figure 4.4, storage could be introduced at (v,t) if $\hat{t}^{k,in}_v \leq t < \bar{t}^{k,in}_v$. Since packet k departs the preceding node at the same time in $\bar{x}$ and $\hat{x}$, it follows that $\bar{t}^{k,in}_v \in \{t+1, t+2, \cdots, \mathtt{n}_S(v,t)-1\}$. Thus each packet in $\mathcal{K}^1_{\hat{x}}(v)$ that introduces storage at (v,t) must have previously been stored at v at one of the times in this interval, which has length $\mathtt{m_S}(v,t)-1$. Therefore, $\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \leq \sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \bar{x}^k_f + (\mathtt{m_S}(v,t)-1)b_v$. □

In the following lemma we observe that this bound can be significantly tightened. This tightening relies on the argument that if two packets j and k in $\mathcal{K}^1_{\hat{x}}(v)$ where not stored at v at the same time in $\bar{x}$, then the same is true in $\hat{x}$.

Lemma 4.5. *If D_S satisfies properties $(P1)$ and $(P2)$, then*

$$\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \leq b_v.$$

If in addition, $\mathtt{n}_S(v,t) = t+1$, then

$$\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \leq \sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \bar{x}^k_f.$$

Proof. Suppose packet k was stored at v in $\bar{x}$ ($\bar{t}^{k,in}_v < \bar{t}^{k,out}_v$). Again, storage could be introduced if $\hat{t}^{k,in}_v < \bar{t}^{k,in}_v$. However, we will show that if commodities k and j in $\mathcal{K}^1_{\hat{x}}(v)$ were not stored at the same time at v according to $\bar{x}$, then the same holds for $\hat{x}$. This would prove that $\sum_{k\in\mathcal{K}^1_{\hat{x}}(v)} \hat{x}^k_e \leq b_v$ since $\bar{x}$ was feasible.

Without loss of generality, we may assume $\bar{t}_v^{k,out} \leq \bar{t}_v^{j,in}$, as in Figure 4.5 ($\bar{t}_v^{k,out} = 3$ and $\bar{t}_v^{j,in} = 3$), since one packet must have departed node v no later than the arrival time of the other. We claim that $\hat{t}_v^{k,out} \leq \hat{t}_v^{j,in}$.

Suppose $\hat{t}_v^{j,in} \geq \bar{t}_v^{k,out}$. Then since $\hat{t}_v^{k,out} \leq \bar{t}_v^{k,out}$ by (F3), it follows that $\hat{t}_v^{k,out} \leq \hat{t}_v^{j,in}$. Alternatively, suppose $\hat{t}_v^{j,in} < \bar{t}_v^{k,out}$. By (F1),

$$\hat{t}_v^{k,out} = \max\{t : t \leq \bar{t}_v^{k,out}, (v,t) \in N_S\}.$$

Let u be the node that packet j visits before v according to $\bar{x}$. By (F2) and since $\hat{t}_u^{j,out} = \bar{t}_u^{j,out}$ (S1),

$$\hat{t}_v^{j,in} = \max\{t : t \leq \hat{t}_v^{j,out} + \tau_{uv} = \bar{t}_v^{j,in}, (v,t) \in N_S\}.$$

Since $\hat{t}_v^{j,in} < \bar{t}_v^{k,out}$ and $\bar{t}_v^{k,out} \leq \bar{t}_v^{j,in}$, it follows that $\hat{t}_v^{k,out} = \hat{t}_v^{j,in} = \max\{t : t \leq \bar{t}_v^{k,out}, (v,t) \in N_S\}$ as required. This proves that $\hat{x}^1$ satisfies storage constraints at all $(v,t) \in N_S$.

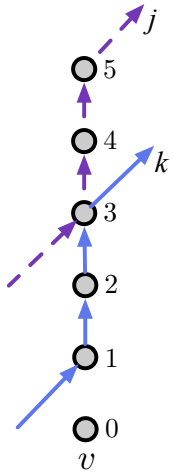


Figure 4.5: Disjoint storage

Finally, when $(v, t+1) \in N_S$, then the set of packets in $\mathcal{K}^1_{\hat{x}}(v)$ stored at (v,t) in $\hat{x}$ are precisely those that are stored at (v,t) in $\bar{x}$. Therefore, if $\mathtt{n}_S(v,t) = t+1$, then $\sum_{k \in \mathcal{K}^1_{\hat{x}}(v)} \hat{x}_e^k \leq \sum_{k \in \mathcal{K}^1_{\hat{x}}(v)} \bar{x}_f^k$. □

We now bound the storage of packets in $\mathcal{K}^2_{\hat{x}}(v)$. In order to be stored at v at time t, a packet must arrive at v by time t. This gives an upper bound on the time the packet could have departed the previous node in D_S, which implies a corresponding upper bound on the time the packet could have departed the previous node in $\bar{x}$. Similarly, we identify a lower bound on the time a packet in $\mathcal{K}^2_{\hat{x}}(v)$ could depart the previous node in D_T if it was not stored at (v,t) in $\bar{x}$, but is stored at (v,t) in $\hat{x}$.

For ease of notation, we introduce an additional definition. Given partially and fully time-expanded networks D_S and D_T respectively, for each $e = ((v,t),(v,t')) \in H_S$ we define

$$U_e(D_S, D_T) := \sum_{(w,t') \in N_T^-(v,t) \cup N_S^-(v,t)} u_{wv} \cdot (\mathtt{m}_S(w,t') - 1).$$

We write U_e when D_S and D_T are self-evident.

Lemma 4.6. *If D_S satisfies properties (P1) and (P2), then*

$$\sum_{k \in \mathcal{K}^2_{\hat{x}}(v)} \hat{x}_e^k \leq \sum_{k \in \mathcal{K}^2_{\hat{x}}(v)} \bar{x}_f^k + U_e.$$

Proof. Let $w \in N^-(v)$. We know that multiple timed arcs of (w, v) in A_T are mapped to the same timed arc $((w, t_1), (v, t_2))$ in A_S as discussed in Section 4.2.1 which could introduce additional storage at v. Throughout this proof we will consider Figure 4.6. The missing nodes are marked with white squares, and the dashed gray arc shows the arc we would obtain if $(w, t - \tau_{wv})$ was in the current partially time-expanded network (which it may or may not be).

Suppose k is a packet in $\mathcal{K}^2_{\hat{x}}(v)$ where w is the node it visits immediately before v according to the solution $\bar{x}$. That is, packet k travels along a timed arc $((w, \bar{t}^{k,out}_w), (v, \bar{t}^{k,in}_v))$ in D_T. First, consider the case where $\bar{t}^{k,in}_v \leq t$. If packet k was originally stored at (v, t) in $\bar{x}$, then the storage needed at (v, t) for packet k in $\hat{x}$ cannot exceed the level in $\bar{x}$. If instead packet k was not stored at v, then $\bar{t}^{k,out}_v \leq t$ and so $\hat{t}^{k,out}_v \leq t$ as well. As a result, packet k would not stored at (v, t) in $\hat{x}$.

Thus, it is only possible to introduce storage at (v, t) in $\hat{x}$ for packet k when $\bar{t}^{k,in}_v > t$ and $\hat{t}^{k,in}_v \leq t$. The first condition is equivalent to $\bar{t}^{k,out}_w > t - \tau_{wv}$.

We now consider the second condition, $\hat{t}^{k,in}_v \leq t$. First observe that if $\bar{t}^{k,out}_w < \mathtt{n_S}(w, t - \tau_{wv})$, then $\hat{t}^{k,in}_v \leq t$. Thus, it could be the case that

$$\bar{t}^{k,out}_w \in \{t - \tau_{wv} + 1, t - \tau_{wv} + 2, \cdots, \mathtt{n_S}(w, t - \tau_{wv}) - 1\}.$$

This interval has length $\mathtt{m_S}(w, t - \tau_{wv}) - 1$, and we mark these departure times with (†) in Figure 4.6. Note that $(w, t - \tau_{wv}) \in N^-_T(v, t)$.

Alternatively we could have $\bar{t}^{k,out}_w \geq \mathtt{n_S}(w, t - \tau_{wv})$, and if $\hat{t}^{k,in}_v \leq t$, property (P2) implies that $\hat{t}^{k,in}_v = t$. That is, $\mu(((w, \bar{t}^{k,out}_w), (v, \bar{t}^{k,in}_v)))$ is an incoming timed arc at (v, t) in D_S. Thus, $(w, \hat{t}^{k,out}_w) \in N^-_S(v, t)$. This set is marked with $(*)$ in Figure 4.6.

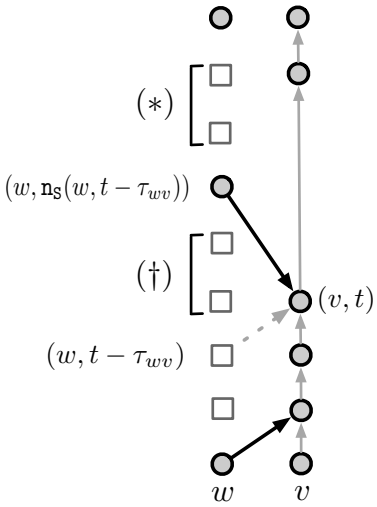


Figure 4.6

Thus, the additional storage needed at (v, t) to accommodate packets in $\mathcal{K}^2_{\hat{x}}(v)$ is at most

$$\sum_{(w,t') \in N^-_T(v,t) \cup N^-_S(v,t)} u_{wv} \cdot (\mathtt{m_S}(w, t') - 1) = U_e.$$

□

Thus, we introduce the following property. The proof of Lemma 4.7 follows from Lemmas 4.3, 4.5, and 4.6 along with the fact that for each $e \in H_S$, $\sum_{k \in \mathcal{K}_v} \hat{x}_e^k = \sum_{k \in \mathcal{K}^1_{\hat{x}}(v)} \hat{x}_e^k + \sum_{k \in \mathcal{K}^2_{\hat{x}}(v)} \hat{x}_e^k$.

($P^{\texttt{storage}}$): For any $e = ((v,t),(v,t')) \in H_S$,

$$b'_e \geq \begin{cases} b_v + U_e & \text{if } (v, t+1) \in N_S \\ 2b_v + U_e & \text{if } (v, t+1) \notin N_S \end{cases}$$

Lemma 4.7. *Let D_S be a partially time-expanded network that satisfies properties $(P1)$, $(P2)$, and $(P^{\texttt{storage}})$, and let $\bar{x}$ be a solution to UPR(D_T). Then $\hat{x} = \mu(\bar{x})$ satisfies constraint (12).*

Finally, we prove the following theorem that UPR(D_S) is indeed a lower bound.

Theorem 4.8. *If D_S satisfies properties $(P1), (P2)$, $(P^{\texttt{arcs}})$, and $(P^{\texttt{storage}})$, then the objective value of an optimal solution to UPR(D_S) is at most the objective value of an optimal solution to UPR(D_T).*

Proof. Let $\bar{x}$ be a solution to UPR(D_T), and let $\hat{x} = \mu(\bar{x})$. By Lemma 4.1, $\hat{x}$ satisfies the flow and integrality constraints of UPR(D_S). By Lemmas 4.2 and 4.7, $\hat{x}$ satisfies the arc capacity and storage capacity constraints of UPR(D_S). Finally, since μ maps trajectories in D_T to trajectories in D_S that underestimate the original length, the objective value of $\hat{x}$ in UPR(D_S) is at most the objective value of $\bar{x}$. □

It is important to note that when $N_S = N_T$, UPR(D_S) is equivalent to UPR(D_T). Furthermore, the objective value of UPR(D_S) is non-decreasing as we add timed nodes to N_S. However, we would ultimately like to solve UPR(D_T) without having to use $N_S = N_T$. In the following sections, we will describe how to detect when a solution to UPR(D_S) can be converted to a solution of UPR(D_T) of equal makespan, and if not, how we select the timed nodes to add to N_S. The minimal set of timed nodes satisfying $(P1), (P2)$, $(P^{\texttt{arcs}})$, and $(P^{\texttt{storage}})$, is the set of all nodes at times 0 and T. We begin the algorithm with this set of timed nodes.

Algorithm 8: `Generate-Initial-`$N_S(D = (N, A), \mathcal{K}, T)$

Input: Base network $D = (N, A)$, packet set $\mathcal{K}$, and upper bound, T, on the optimal makespan
$N_S \leftarrow \emptyset$
for $v \in N$ **do**
    $N_S \leftarrow N_S \cup (v, 0) \cup (v, T)$
return N_S

In Algorithm 9 we take as input the current set of timed nodes and generate the timed arcs A_S and H_S along with capacities u' and b' so that $D_S = (N_S, A_S \cup H_S), u', b'$ satisfies $(P1), (P2)$, $(P^{\texttt{arcs}})$, and $(P^{\texttt{storage}})$. We define the capacities so that they satisfy $(P^{\texttt{arcs}})$ and $(P^{\texttt{storage}})$ with equality.

Algorithm 9: `Generate-`$A_S \cup H_S(N_S,\ D, T)$

Input: Base network $D = (N, A)$, and a set of timed nodes, N_S
for $(v, t) \in N_S$ **do**
    $e \leftarrow ((v, t), (v, \mathtt{n_S}(v, t)))$
    $U_e(D_S, D_T) \leftarrow \sum_{(w,t') \in N_T^-(v,t) \cup N_S^-(v,t)} u_{wv} \cdot (\mathtt{m_S}(w, t') - 1)$
    $b'_e := \begin{cases} b_v + U_e(D_S, D_T) & \text{if } (v, t+1) \in N_S \\ 2b_v + U_e(D_S, D_T) & \text{if } (v, t+1) \notin N_S \end{cases}$
    If $t < T$, add timed arc e to H_S with storage capacity b'_e
    forall $vw \in A$ **do**
        Add timed arc $f = ((v, t), (w, t'))$ with capacity $u'_f = u_{vw} \cdot \mathtt{m_S}(v, t)$ to A_S where t' is the largest value such that $(w, t') \in N_S$ and $t' \leq t + \tau_{vw}$
return A_S, H_S

In Section 4.4.2 we observe that the DDD algorithm performs comparatively better on sparse instances – instances where each node has low degree in the flat network. This is perhaps unsurprising given the nature of the relaxation for node storage given a partial network $D_S = (N_S, A_S)$. While the relaxed arc capacity on a timed arc $((v, t), (w, t'))$ only relies on the timed copies of v included in N_S, the relaxed node storage capacity at a timed node (v, t) relies on the timed copies of v in N_S, as well as the timed copies of each node in $N_D^-(v)$. When the graph is sparse this set of incoming neighbours is small for each node and fewer timed nodes must be added in the augmentation procedure.

4.3 Upper bound model and augmentation

In this section we define the upper bound and refinement procedures, and state the full DDD algorithm for UPR.

4.3.1 Upper bound model

Given a solution to the lower bound model, we want to determine if it can be converted to an optimal solution of UPR(D_T). Suppose we are working with a partially time-expanded network $D_S = (N_S, A_S \cup H_S)$ that satisfies (P1), (P2), ($P^{\texttt{arcs}}$), and ($P^{\texttt{storage}}$). Let $\hat{x}$ be an optimal solution to UPR(D_S) with value $\hat{T}$. We would like to know if $\hat{x}$ can be converted to a solution to UPR(D_T) with the same value (i.e. makespan).

Observe that $\hat{x}$ specifies a trajectory in D_S for each packet, each of which corresponds to a path in the underlying static graph D. Additionally, we are given a candidate makespan $\hat{T}$. Thus, we can generate an upper bound for UPR(D_T) if we solve UPR in D_T with the added the restriction that packets follow the underlying paths in D specified by $\hat{x}$. Specifically, we have an instance of UPR-FP where for each $k \in \mathcal{K}$, P_k is the path in D induced by $\hat{x}$ for packet k. Let A_T^k, and H_T^k denote the set of timed arcs that could be used for a trajectory with underlying path P_k. That is, for each $k \in \mathcal{K}$,

$$A_T^k = \{((v,t),(w,t')) \in A_T : vw \in A(P_k)\} \quad \text{and} \quad H_T^k = \{((v,t),(v,t')) \in H_T : v \in N(P_k)\}.$$

While this instance of UPR-FP can still be solved more quickly than the original UPR instance with the same time horizon T, it is still NP-hard [48]. Thus, we forfeit the ability to obtain an upper bound in each iteration, and instead we will restrict the time horizon to be $T' = \lceil (1+\delta)\hat{T} \rceil$ for some $\delta \geq 0$ (we used $\delta = 0.01$ for our computations). This restriction of the time horizon allows us to detect if $\hat{x}$ can be converted to a solution to UPR(D_T) with value at most T'. Let $D_{T'} = (N_{T'}, A_{T'} \cup H_{T'})$ be the fully time-expanded network with time horizon T'. We now present the following upper bound formulation. Note that the arc capacities and node storage levels match the values given in the original UPR(D_T) instance.

$$\begin{aligned}
\min \quad & \bar{T} && \text{(UPR-FP}(\{D^k_{T'}\}_{k\in\mathcal{K}}, D_{T'}))\\
\text{s.t.} \quad & t' \cdot x^k_e \le \bar{T} \quad \forall k \in \mathcal{K},\ \forall e = ((v,t),(w,t')) \in A^k_{T'} && (4.13)\\
& \sum_{e=((v,t),(w,t'))\in A^{k,final}_{T',k}} t' \cdot x^k_e \le \bar{T} \quad \forall k \in \mathcal{K} && (4.14)\\
& \sum_{e\in\delta^+_{D^k_{T'}}(v,t)} x^k_e - \sum_{e\in\delta^-_{D^k_{T'}}(v,t)} x^k_e = \begin{cases} 1 & (v,t) = (s_k, 0)\\ -1 & (v,t) = (t_k, T')\\ 0 & \text{otherwise}\end{cases} \quad \forall k \in \mathcal{K}, (v,t) \in N_{T'} && (4.15)\\
& \sum_{k\in\mathcal{K}} x^k_e \le u_e \quad \forall e \in A_{T'} && (4.16)\\
& \sum_{k\in\mathcal{K}_v} x^k_e \le b_e \quad \forall e \in H_{T'} && (4.17)\\
& x^k_e \in \{0,1\} \quad \forall k \in \mathcal{K}, e \in A^k_{T'} \cup H^k_{T'}. && (4.18)
\end{aligned}$$

This gives the following upper-bound procedure. Algorithm 10 takes as input an optimal solution $\hat{x}$ to the current partially time-expanded network with makespan $\hat{T}$. We also take as input the optimality factor tolerance $\delta \ge 0$ and the current best-known upper bound on T^*, denoted `UB`. $\hat{x}$ defines a trajectory $\hat{Q}_k$ for each $k \in \mathcal{K}$, and projecting this down to the base graph defines a path P_k for each $k \in \mathcal{K}$. We then solve the UPR problem with fixed paths with an upper bound of $T' = \lceil (1+\delta)\hat{T} \rceil$, where each $k \in \mathcal{K}$ must follow a trajectory with underlying path P_k. If the problem is feasible, the we check if the value V is less than our current best upper bound and output the current feasible solution. Otherwise we return the original upper bound.

Algorithm 10: `Compute-UB`$(D, \hat{x}, \hat{T}, \delta, \beta)$

Input: Base network $D = (N, A)$, an optimal partial network solution $\hat{x}$ with value $\hat{T}$, and parameter $\delta \geq 0$, and a current upper bound on the value of T^*, denoted β
$T' = \lceil (1+\delta)\hat{T} \rceil$, and $\bar{x} = \emptyset$
$D_{T'} = (N_{T'}, A_{T'} \cup H_{T'})$
Let $\hat{Q} = \{\hat{Q}_k\}_{k \in \mathcal{K}}$ denote the set of trajectories given by $\hat{x}$
For each $k \in \mathcal{K}$ let P_k denote the underlying path in D of trajectory $\hat{Q}_k$
$A^k_{T'} = \{((v,t),(w,t')) \in A_{T'} : vw \in A(P_k)\}$
$H^k_{T'} = \{((v,t),(v,t')) \in H_{T'} : v \in N(P_k)\}$
$D^k_{T'} = (N_{T'}, A^k_{T'} \cup H^k_{T'})$
Solve UPR-FP$(\{D^k_{T'}\}_{k \in \mathcal{K}}, D_{T'})$
if *UPR-FP*$(\{D^k_{T'}\}_{k \in \mathcal{K}}, D_{T'})$ *is feasible and has optimal value* $\bar{T}'$ **then**
    Let $\bar{x}$ be an optimal solution to UPR-FP$(\{D^k_{T'}\}_{k \in \mathcal{K}}, D_{T'})$
    `UB` $= \min\{\beta, \bar{T}'\}$
else
    `UB` $= \beta$
return `UB`, $\bar{x}$

4.3.2 Augmentation step

We now consider the case where the partial solution $\hat{x}$ cannot be converted to a solution of UPR(D_T) of equal cost using the upper bound model. In this section, we will detail how to augment the set N_S.

Due to our relaxation procedure, we know that $\hat{x}$ may not be convertible to a solution to UPR(D_T) with equal makespan due to shortened arcs in D_S, relaxed arc capacities, and relaxed node storage levels.

Let $e = ((v,t),(w,t')) \in A_S \cup H_S$ be a timed arc in the support of $\hat{x}$. There is a well-established method to correct short arcs in A_S [2]:

$$\text{if } t' < t + \tau_{vw}, \text{ we add the timed node } (w, t + \tau_{vw}).$$

We now proceed to deal with arcs exceeding arc and storage capacities. For each timed arc $e = ((v,t),(w,t')) \in A_S \cup H_S$, let $\hat{x}_e$ be the total active flow assigned to arc e according to

$\hat{x}$. That is,

$$\hat{x}_e = \begin{cases} \sum_{k \in \mathcal{K}} \hat{x}_e^k & e \in A_S \\ \sum_{k \in \mathcal{K}_v} \hat{x}_e^k & e \in H_S \end{cases}$$

If $e \in A_S$ and $\hat{x}_e > u_e$, then by construction of u', $(v, t+1) \notin N_S$. Thus, we will add $(v, t+1)$ to N_S.

If instead $e \in H_S$ and $\hat{x}_e > b_e$, by definition of b'_e and $\mathtt{m}_S(v,t)$, it follows that for some $(z, \bar{t}) \in N_T^-(v,t)$ or $(z, \bar{t}) \in N_S^-(v,t)$, we have $\mathtt{m}_S(z, \bar{t}) > 1$. For each $(z, \bar{t}) \in N_S^-(v,t)$ with $\mathtt{m}_S(z, \bar{t}) > 1$, we add $(z, \bar{t}+1)$ to N_S. We also add $(v, t+1)$ to N_S if it is not yet in the set, which then ensures $N_S^-(v,t) \subseteq N_T^-(v,t)$ in the next iteration.

Algorithm 11 on the following page restates each of these procedures.

Algorithm 11: `Augment-`$N_S(D_S, D, \mathcal{K}, \hat{x})$

Input: Current partially time-expanded network $D_S = (N_S, A_S \cup H_S)$, base graph $D = (N, A)$, packet set $\mathcal{K}$, and an optimal solution $\hat{x}$ to UPR(D_S)
$N'_S \leftarrow N_S$
for $e = ((v,t),(w,t')) \in supp(\hat{x}) := \{e \in A_S \cup H_S : \hat{x}_e > 0\}$ **do**
  if $e \in A_S$ **then**
    Compute the flow assigned to timed arc e according to $\hat{x}$, $\hat{x}_e := \sum_{k \in \mathcal{K}} \hat{x}_e^k$.
    if $t' < t + \tau_{vw}$ **then**
      $N'_S \leftarrow N'_S \cup \{(w, t + \tau_{vw})\}$
    if $\hat{x}_e > u_e$ **then**
      $N'_S \leftarrow N'_S \cup \{(v, t+1)\}$.
  if $e \in H_S$ **then**
    Compute the relevant flow assigned to timed arc e according to $\hat{x}$, $\hat{x}_e := \sum_{k \in \mathcal{K}_v} \hat{x}_e^k$.
    if $\hat{x}_e > b_v$ **then**
      for $(z, \bar{t} = t - \tau_{z,v}) \in N_T^-(v,t) : \mathtt{m}_S(z, \bar{t}) > 1$ **do**
        $N'_S \leftarrow N'_S \cup \{(z, \bar{t}+1)\}$.
      $N'_S \leftarrow N'_S \cup \{(v, t+1)\}$.
return N'_S

Proposition 4.9. *Given an instance of UPR with minimum makespan T^*, Algorithm 11 only adds timed nodes (v,t) to N_S with $t \leq T^* + 1$.*

Proof. In any iteration, the optimal solution $\hat{T}$ of UPR(D_S) is at most T^* since UPR(D_S) is a relaxation of UPR(D_T). When correcting an arc $e = ((v,t),(v,t')) \in H_S$ due to exceeded storage capacity, we know that $t' \leq T^*$, since v is not the destination for the commodities contributing to $\hat{x}_e$, and by constraint (4.7). Furthermore, we add nodes (w,t) with $t \leq t'+1$ for this correction since $\tau \geq 0$.

Now consider the correction of an arc $((v,t),(w,t'))$ in A_S. If the arc exceeds capacity u, then we add node $(v,t+1)$ to N_S. Since $t \leq T^*$, clearly $t+1 \leq T^*+1$. Finally, if the arc is too short, then we add the node $(w,t+\tau_{vw})$ to N_S. Due to our replacement of constraint (4.1) with constraint (4.7), we see that $t+\tau_{vw} \leq \hat{T} \leq T^*$. □

Following along the lines of the proof, we easily obtain Corollary 4.10.

Corollary 4.10. *Given an instance of UPR with minimum makespan T^* with $\tau_a > 0$ for all $a \in A$, Algorithm 11 only adds timed nodes (v,t) to N_S with $t \leq T^*$.*

Proposition 4.9 points to the strength of the DDD approach over solving UPR(D_T) when the upper bound T given ends up being much larger than T^*. The DDD approach will maintain a much smaller time-expanded network throughout the algorithm.

In the original application of DDD to SND [2], the solution to the upper bound model dictated which timed arcs were to be corrected in the augmentation step. However, in our model, we correct every timed arc in the support of the optimal solution to UPR(D_S) that is too short, or has exceeded the original arc and storage capacities. As a result, it is not necessary to run the upper bound procedure in each iteration, and instead it may save time to only run the procedure when the makespan reported by two consecutive iterations is similar. While we solved the upper bound model in each iteration in our experiments, it would be worthwhile testing this alternative approach.

Algorithm 12: UPR-DDD$(D, \mathcal{K}, T, \delta)$

Input: Base network $D = (N, A)$, commodity set $\mathcal{K}$, an upper bound, T, on the optimal makespan, and an optimality parameter $\delta \geq 0$
$N_S \leftarrow$ Generate-Initial-$N_S(D, \mathcal{K}, T)$
$\bar{x} = \emptyset$
UB $\leftarrow T$
LB $\leftarrow 0$
gap $= (\texttt{UB} - \texttt{LB})/\texttt{UB}$
while *gap* $> \delta$ *or* $\bar{x} \neq \emptyset$ **do**
    $A_S, H_S \leftarrow$ Generate-$A_S \cup H_S(N_S, D, T)$
    $D_S \leftarrow (N_S, A_S \cup H_S)$
    Solve UPR(D_S), and let $\hat{x}$ be an optimal solution, with value $\hat{T}$
    LB $\leftarrow \max\{\texttt{LB}, \hat{T}\}$
    UB, $\bar{x} \leftarrow$ Compute-UB$(D, \hat{x}, \hat{T}, \delta, \texttt{UB})$
    gap $= (\texttt{UB} - \texttt{LB})/\texttt{UB}$
    if *gap* $\leq \delta$ *and* $\bar{x} \neq \emptyset$ **then**
        Stop. An solution within δ of optimal has been found for UPR(D_T).
        return $\bar{x}$, UB
    else
        $N_S \leftarrow$ Augment-$N_S(D_S, D, \mathcal{K}, \hat{x})$

We now prove correctness of our algorithm as well as bound the number of iterations. It is important to note that we can bound the number of iterations in terms of T^* and not just T.

Theorem 4.11. *The algorithm UPR-DDD$(D, \mathcal{K}, T, \delta)$ terminates with solution that has makespan at most an $(1 + \delta)T^*$in at most $|N|T^*$ iterations.*

Proof. First recall that since all input data is integral, and we are given that $T^* \leq T$, the decision times of an optimal solution are in $[T]$.

Consider an iteration of the algorithm where the partially time-expanded network is $D_S = (N_S, A_S \cup H_S)$ and the relaxed capacities are given by u' and b'. Let $\hat{Q}$ be the set of trajectories in D_S that gives a min makespan routing. For each $k \in \mathcal{K}$, let $\hat{Q}_k$ denote the trajectory for packet k in $\hat{Q}$. Let $\hat{T}$ denote the makespan of $\hat{Q}$ in D_S. Let $\bar{Q}$ be the set of trajectories we obtain by solving the corresponding upper bound, and suppose the factor

gap between the two makespans is greater than δ.

It follows that we could not obtain trajectories in D_T with the same underlying paths as $\hat{Q}$ while satisfying the original capacities u and b, given a time horizon of $(1+\delta)\hat{T}$. Specifically, it must have been infeasible to simply assign each packet the trajectory $\hat{Q}_k$ in D_T. Thus, it must be that some timed arc in $\hat{Q}$ was too short, or exceeded the arc capacity or node storage level. Therefore, in each scenario there must have been a timed node $(v,t) \in N_T \setminus N_S$ that we can add to N_S.

Furthermore, we proved in Proposition 4.9 that our algorithm only adds timed nodes (v,t) to N_S with $t \leq T^* + 1$. Since in each iteration we add at least one timed node and in the first iteration we have at least one copy of each node, the DDD algorithm terminates within $|N|T^*$ iterations. □

Corollary 4.12. *The algorithm* UPR-DDD$(D, \mathcal{K}, T, \delta = 0)$ *terminates with an optimal solution in at most* $|N|T^*$ *iterations.*

4.4 Computational results

To demonstrate the effectiveness of our DDD algorithm, we compare the runtime of the DDD algorithm and the original full integer program UPR(D_T) when applied to geographic and geometric instances. For the geographic instances, we base the node and arc selection on the population centres in the United States. Our geometric instances are constructed to model social networks.

For each problem instance, we initially solve the DDD instance with a sufficiently large time horizon T so that $T^* < T$. This initial solve gives us the value of T^*. Then to compare the solve time for DDD and the full IP, we run each algorithm with the time horizon upper bound of $T^*, 1.5T^*$, and $2T^*$ for up to two hours. Thus, in total we solve each instance seven times. Note, we use upper bounds T as factors of T^* only for analysis purposes. In practice, we would select a value of T that is sufficiently large so that all packets could be routed within time T. Each algorithm was coded in Python 3.6.9 with Gurobi 8.1.1 [26] as the optimization solver. The running time limit was set to 7200 seconds (two hours) using the deterministic option of the solver and the instances are solved to within 1% of optimality. The instances were run in a 64 cores 2.6GHz Xeon Gold 6142 Processor with 256GB RAM, running a Linux operating system. Each instance was run with a limit of 5 cores. The generated instances can be found at https://github.com/madisonvandyk/UPRlib.

4.4.1 Geographic instances

Dataset

For the base graph, we use the locations of the top n most populated cities in the USA. We randomly select m arcs to form A, and set τ_a to be the distance in hundreds of miles, rounded up to the nearest integer. We compute the shortest directed path between each pair of vertices. We then select k random origin-destination pairs from the digraph D such that there is dipath from the origin to the destination, and the shortest path has at least δ arcs and length at most a factor γ times the max shortest path length of any pair. We construct the origin-destination pairs in this way to ensure that min makespan is not simply the max length of the shortest path. For arc and node capacities, we follow a discrete version of the approach of Crainic et al. [10, 11] that was developed as a rigorous test set for SND. This dataset construction has since been modified to analyze the performance of DDD algorithms [2, 39], which were the instances used in Chapter 3, Section 3.4. We restate the overall construction here.

We select capacities from a discrete uniform distribution with endpoints $[\alpha_1, \alpha_2]$ and $[\beta_1, \beta_2]$ for node storage. Crainic et al. [10, 11] introduced the *capacity ratio* $C = |A|k/\sum_{e\in A} u_e$. As C approaches 1, the network is lightly capacitated, and the congestion level increases as C increases. Crainic studied scenarios with $C \in \{1, 2, 8\}$ when solving SND. However, since in UPR we are not incentivized to consolidate packet flow as is the case of SND, we need much more restrictive congestion to generate problems of interest (minimal congestion would allow all packets to be routed along shortest paths, using no node storage). We now list our set of parameters.

Parameters

- $n = 20$ – number of nodes;
- $m \in \{30, 45, 60\}$ – number of arcs;
- $k \in \{200, 250, 300\}$ – number of packets;
- $(\alpha_1, \alpha_2) \in \{(1, \lceil 0.01k \rceil), (1, \lceil 0.0175k \rceil), (1, \lceil 0.025k \rceil)\}$ – bounds for arc capacity;
- $(\beta_1, \beta_2) \in \{(0, \lceil 0.01k \rceil), (0, \lceil 0.0175k \rceil), (0, \lceil 0.025k \rceil)\rceil\}$ – bounds storage capacity;
- $\delta = 3, \gamma = 0.90$.

The choices for α_1, α_2, and β_1, β_2 allow for a range of congestion levels, while ensuring that the resulting instance is always feasible. The graph sparsity levels and capacity levels relative to the number of commodities were chosen by running computational tests to ensure that the capacities increased the optimal makespan over the uncapacitated instances. The three values of m are selected to ensure we examine the DDD algorithm on networks of

varying connectivity, while maintaining that the instances are capacitated – overly dense graphs would allow for many vertex pairs to be connected via few arcs, decreasing the congestion. Similarly, the range of values of k allows us to analyze the effectiveness of our DDD algorithm on various densities.

We offer a quick overview of how the above parameters impact the optimal time horizon T^* as well as the overall solve time for UPR(D_T), holding all other parameters constant. As m increases, T^* decreases since packets can travel via shorter direct paths, and fewer packets are forced to overlap. While larger m would imply that the UPR(D_T) takes longer to generate, since T^* decreases significantly the overall solve time decreases in our experiments. Naturally, as k increases, T^* increases. As expected, as α and β increase, T^* decreases.

Results

We first present the average runtime (in seconds) among all settings of α, β, when T, m, and k are fixed. We note that the "ratio" column denotes the average ratio of the runtimes, rather than the ratio of the average runtimes. Averages marked with * indicate that there was at least one instance that did not terminate within the time limit.

	$k = 200$			$k = 250$			$k = 300$		
UB	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio
T^*	267	878	2.82	288	693	2.02	674	1,463*	1.98
$1.5T^*$	709	1,293	1.85	1,582	1,135	0.67	1,761	1,602*	0.88
$2T^*$	1,891*	959*	0.85*	2,714*	1,368*	0.42*	2,879*	1,838*	0.51*

Table 4.1: Average runtime for $m = 30$.

	$k = 200$			$k = 250$			$k = 300$		
UB	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio
T^*	144	303	2.08	210	338	1.73	219	345	1.68
$1.5T^*$	406	467	1.25	882	734	0.93	975	814	0.79
$2T^*$	1,491	580	0.40	2,232	652	0.34	3,100*	1,024	0.42*

Table 4.2: Average runtime for $m = 45$.

	$k = 200$			$k = 250$			$k = 300$		
UB	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio
T^*	135	315	2.25	152	320	2.13	189	467	2.44
$1.5T^*$	390	517	1.41	485	652	1.27	800	1,958	2.04
$2T^*$	1,374	532	0.55	1,513	726	0.55	2,814*	2,162*	0.76*

Table 4.3: Average runtime for $m = 60$.

Across all scenarios we see a clear trend that as the upper bound T increases relative to T^*, the increase to the runtime to UPR(D_T) is much greater than the increase to the runtime of DDD. This result is intuitive, since as T increases, UPR(D_T) becomes larger and additional symmetries are introduced in the network – most packets will have an increasing number of possible trajectories in an optimal solution. Since the partially time-expanded networks are sparse, many of these additional symmetries are avoided.

Figures 4.7 - 4.12 report the runtime of the experiments when $T = 1.5T^*$ and $T = 2T^*$. Each plot presents the number of instances solved within a given time limit. These figures demonstrate the same findings as the tables above. The performance of DDD becomes increasingly advantageous over the full IP as m decreases and T increases.

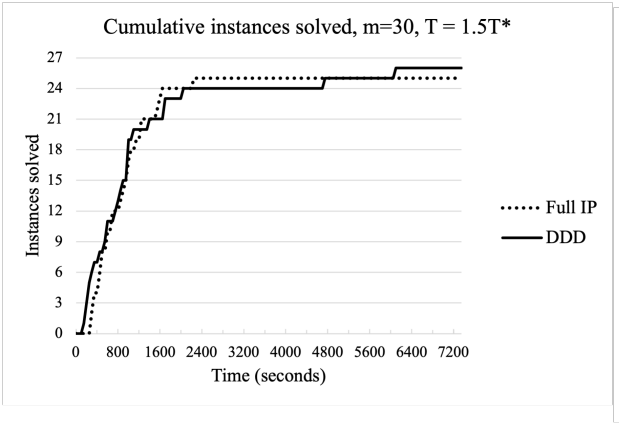


Figure 4.7: $m = 30, T = 1.5T^*$

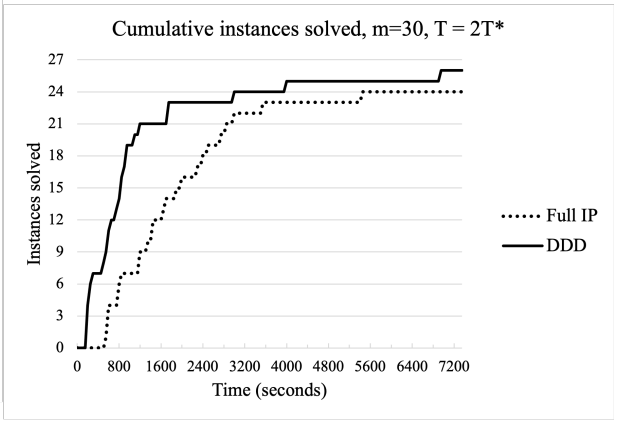


Figure 4.8: $m = 30, T = 2T^*$

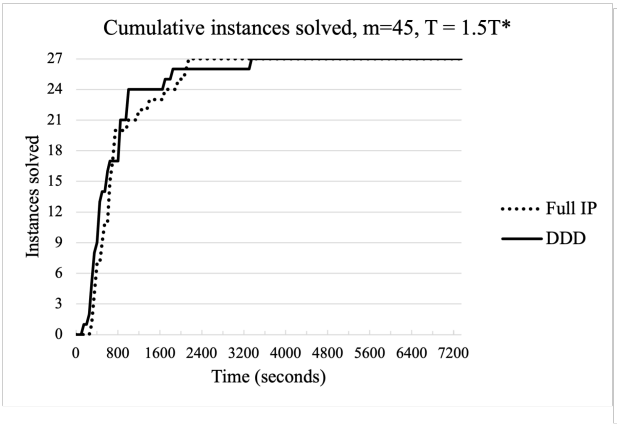


Figure 4.9: $m = 45, T = 1.5T^*$

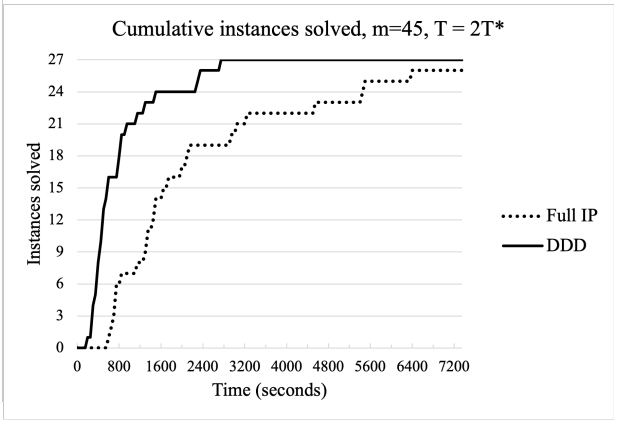


Figure 4.10: $m = 45, T = 2T^*$

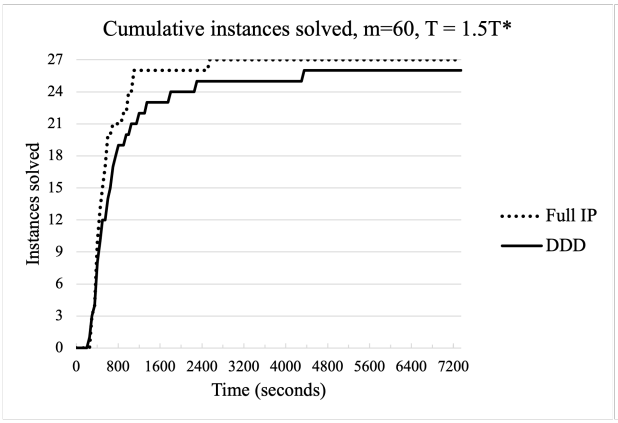


Figure 4.11: $m = 60, T = 1.5T^*$

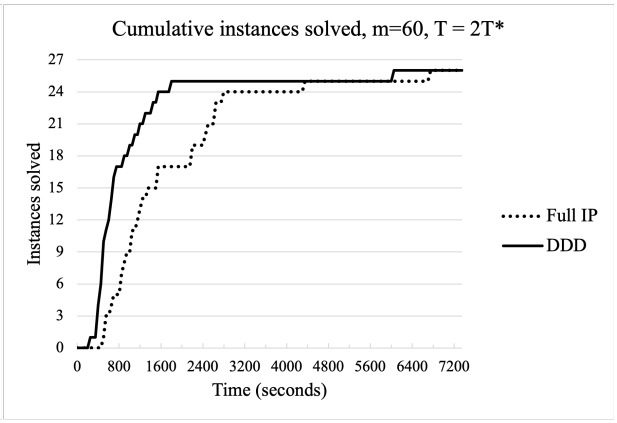


Figure 4.12: $m = 60, T = 2T^*$

Iteration sizes

When $T = T^*$, UPR(D_T) performs better than DDD. This is not surprising since while DDD solves smaller IPs than UPR(D_T), we still require reasonably dense partially time-expanded networks in order for the algorithm to terminate. The following table presents the average number of iterations required until DDD terminates, as well as the average size of the final timed node set compared to the full timed node set.

	$m = 30$		$m = 45$		$m = 60$	
factor	iterations	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$	iterations	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$	iterations	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$
T^*	9.67	0.43	7.00	0.60	6.93	0.72
$1.5T^*$	9.85	0.31	6.93	0.40	6.78	0.47
$2T^*$	9.89	0.23	7.07	0.31	7.07	0.37

Table 4.4: Average number of iterations, and relative size of $N_S^{\texttt{final}}$.

The average number of iterations decreases as the number of arcs, m, increases. This is not surprising since a larger number of arcs allows for packet trajectories with fewer timed arcs, and thus DDD generates feasible trajectories in fewer iterations. At the same time, the average ratio of $|N_S^{\texttt{final}}|/|N_T|$ increases with m, which explains why we do not see better performance for DDD for higher values of m. One reason this ratio is higher is due to the refinement process for storage capacity, since when correcting exceeded storage at a node v, the number of timed nodes added partly depends on the degree of v. We explore the impact of sparsity on the performance of DDD further in Section 4.4.2.

Refinement trends

In each iteration we solve the integer program defined on the partially time-expanded network and obtain a solution $\hat{x}$. If $\hat{x}$ cannot be converted to an optimal solution in D_T, there must be timed arcs in the support of $\hat{x}$ that are either too short, or exceed the original throughput or storage levels. In the following figures, we examine how each of these violated constraint types influences the refinement process in each iteration.

In Figure 4.13 we present the average proportion of the infeasible arcs in the support of $\hat{x}$ that are short, have exceeded throughput, or exceeded storage in each iteration. In Figure 4.14, we present the average proportion of the timed nodes added to correct each violated constraint type in each iteration. We included the test instances where DDD took at least 7 iterations before terminating, and only looked at the first 7 iterations. We see below that this turns out to be sufficient to observe clear trends.

Observe that there are no timed arcs in the support of $\hat{x}$ exceeding storage capacity in the first iteration. This is not surprising since no storage is necessary if there are no (or at least no restrictive) throughput capacities. For the same reason, it is natural that there would be more timed arcs with exceeded throughput than timed holdover arcs with exceeded storage in each iteration. As the iterations progress, the support of $\hat{x}$ increases in size and spreads the flow of packets along a larger number of timed arcs. Thus, it is natural that the proportion of infeasible timed arcs with exceeded throughput decreases. In Figure 4.20,

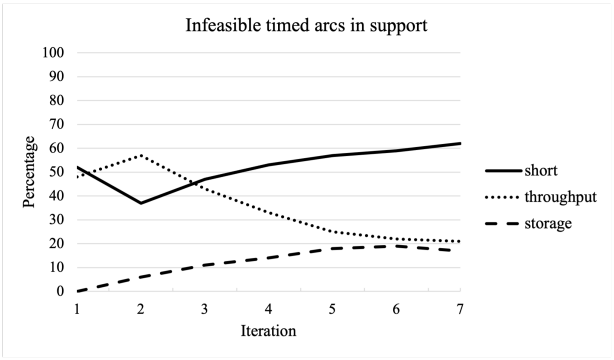


Figure 4.13: Infeasible timed arcs.

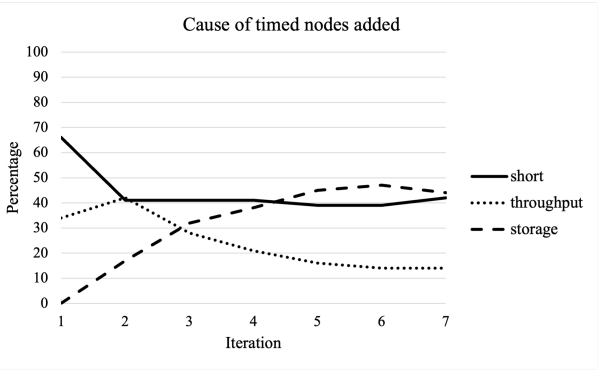


Figure 4.14: Cause of adding timed nodes.

we see that the proportion of timed nodes added due to exceeded storage overtakes those added due to exceeded throughput. This is due to the fact that when correcting exceeded throughput, we add a single timed node, whereas when correcting exceeded storage at a node v, we may add up to $\deg_D(v) + 1$ timed nodes.

4.4.2 Geometric instances

Dataset

Our instances consist of random geometric graphs which have been used widely to model human social networks [33, 46]. We begin by selecting n nodes randomly from an $l \times l$ grid. We then connect each ordered pair of nodes with an arc if their L1-norm distance is at most p. This is a discrete version of a random geometric graph. Since social networks have low diameter, we augment our arc set according to the popular construction of Kleinberg [33]. That is, for each of the nodes $v \in N$, we add q "long-range" arcs (v, w) chosen independently at random, where the ith directed arc from v has endpoint w with probability proportional to $||v - w||_1^{-r}$. For each of the arcs generated to form A, we assign the transit time to be equal to the L1-norm distance between the endpoints.

We then select k random origin-destination pairs from the digraph D according to the same process as in the geographic instances. As was the case of our geographic instances, we select capacities from a discrete uniform distribution with endpoints (α_1, α_2) and (β_1, β_2) for arc capacity and node storage respectively. We now list our set of parameters.

Parameters

- $l = 25$ – grid length and width;
- $n = 20$ – number of nodes;
- $k \in \{200, 225, 250\}$ – number of packets;
- $p \in \{3, 4\}$ – radius for local connections;
- $q \in \{1, \{1, 2\}\}$ – number of long-range connections for each node;
- $r = 0.5$ – scaling factor to select long-range connections;
- $(\alpha_1, \alpha_2) \in \{(1, \lceil 0.01k \rceil), (1, \lceil 0.02k \rceil)\}$ – bounds for arc capacity;
- $(\beta_1, \beta_2) \in \{(0, \lceil 0.01k \rceil), (0, \lceil 0.02k \rceil)\rceil\}$ – bounds storage capacity.

When $q = \{1, 2\}$, for each node we select 1 or 2 long distance arcs with equal probability. We select p and q to be sufficiently small so that the instance is capacitated while still allowing for differing levels of local and global connectivity. In Figure 4.15 we see a sparse network obtained using n and l as stated, with $r = 0.5$, $p = 3$, and $q = 1$. In contrast, Figure 4.16 shows a dense network obtained with parameters $r = 0.5$, $p = 4$, and $q = \{1, 2\}$. In each figure the placement of the nodes corresponds to the location in the $l \times l$ grid.

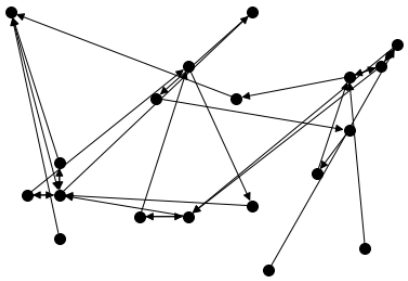


Figure 4.15: Locally and globally sparse

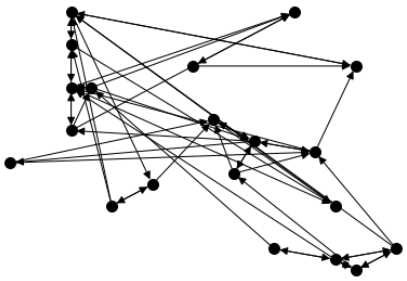


Figure 4.16: Locally and globally dense

When $r = 0$, the long-range connections are chosen uniformly at random, and when $r = 1$, the connections are chosen with probability proportional to the inverse distance. Since we would like some level of global connectivity, but long-distance connections are less common than short distance connections in social networks, we use $r = 0.5$. As stated, we require higher congestion in UPR than in SND to generate interesting problems. The choices for α_1, α_2, and β_1, β_2 allow for a range of congestion levels, while ensuring that the resulting instance is feasible.

We offer a quick overview of how the above parameters impact the optimal time-horizon T^* as well as the overall solve time for UPR(D_T), holding all other parameters constant. As

p and q increase, T^* decreases since packets can travel via shorter direct paths, and fewer packets are forced to overlap. As the arc and storage capacities increase, T^* decreases since the network can accommodate a higher number of active packets.

Results

Overall, we see that DDD is faster than solving the full IP when $T = 2T^*$, and faster for slow sparse instances when $T = 1.5T^*$. On average, we find that when $T = 2T^*$, DDD completes in 49% of the time it takes to run UPR(D_T). Note, this is an *overestimate* since when running UPR(D_T), some of the instances did not complete within the allowed time.

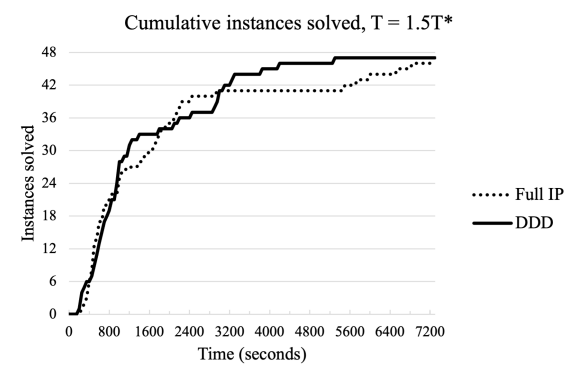


Figure 4.17: Cumulative solved, $T = 1.5T^*$

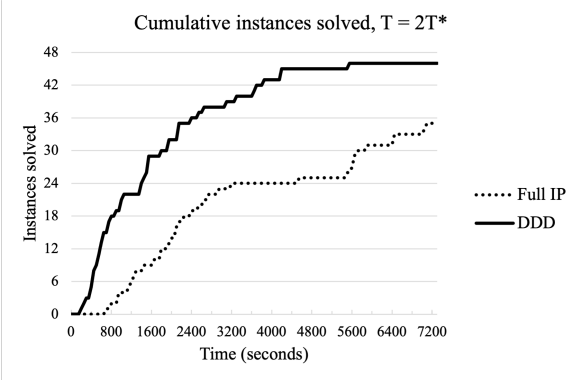


Figure 4.18: Cumulative solved, $T = 2T^*$

To examine the results more closely, we divide the instances into groups based on the level of local and global connectivity of the underlying network. Again, the "ratio" column given is the average of the ratio between the runtime of UPR(D_T) and DDD. Again, averages marked with * indicate that there was at least one instance that did not terminate within the time limit. As a result, the ratio in the same row could be lower if all instances were run until termination.

Low local and low global connectivity

	$k = 200$			$k = 225$			$k = 250$		
UB	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio
T^*	363	1,946	6.17	477	1,069	2.17	651	1,921	4.41
$1.5T^*$	2,705	3,220	1.18	5,066*	2,702	1.15*	3,644	1,523	0.47
$2T^*$	6,780*	3,084	0.44*	6,191*	3,188	0.79*	5,791*	1,663	0.27*

Table 4.5: Average runtime, $(p = 3, q = 1)$.

Low local and high global connectivity

	$k = 200$			$k = 225$			$k = 250$		
UB	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio
T^*	234	821	3.20	285	550	2.97	245	893	4.09
$1.5T^*$	1,205	768	0.87	2,465*	1,235	2.13*	1,029	2,672	3.61
$2T^*$	2,803	2,904*	0.75	3,591*	1,391	0.59*	6,284*	1,584	0.37*

Table 4.6: Average runtime, $(p = 3, q = \{1, 2\})$.

High local and low global connectivity

	$k = 200$			$k = 225$			$k = 250$		
UB	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio
T^*	186	478	2.48	248	734	2.76	281	791	2.70
$1.5T^*$	1,343	1,581	1.48	1,092	1,263	1.23	1,728	1,687	1.05
$2T^*$	4,726*	2,538*	0.40*	3,776*	1,867	0.49*	5,926*	1,880	0.40*

Table 4.7: Average runtime, $(p = 4, q = 1)$.

High local and high global connectivity

	$k = 200$			$k = 225$			$k = 250$		
UB	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio	UPR(D_T)	DDD	ratio
T^*	152	318	2.23	187	382	2.12	202	438	2.20
$1.5T^*$	868	533	0.82	796	539	0.82	590	834	1.41
$2T^*$	2,502	1,093	0.58	3,925*	624	0.31*	1,803	896	0.52

Table 4.8: Average runtime, $(p = 4, q \in \{1, 2\})$.

Overall, we draw the same conclusions as in Section 4.4.1. As the upper bound increases relative to T^*, the performance of DDD improves over the the full integer program. For sparse instances, DDD outperforms UPR(D_T) even when $T = 1.5T^*$ (DDD terminates in an average of 93% of the time of the full IP). In the following table we see that when

the underlying graph is sparser, this increases the average number of iterations until DDD terminates ("iter." column). Despite this increase in average iteration count, the average ratio of $|N_S^{\texttt{final}}|/|N_T|$ decreases and so DDD still performs comparatively better on sparser instances. A future direction of research would be to examine how local and global connectivity impacts the performance of the DDD algorithm in general.

	$p=3,\ q=1$		$p=3,\ q=\{1,2\}$		$p=4,\ q=1$		$p=4,\ q=\{1,2\}$	
factor	iter.	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$	iter.	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$	iter.	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$	iter.	$\lvert N_S^{\texttt{final}}\rvert/\lvert N_T\rvert$
T^*	9.75	0.57	8.42	0.64	7.33	0.64	6.17	0.66
$1.5T^*$	8.42	0.37	8.58	0.45	7.50	0.44	6.25	0.46
$2T^*$	8.58	0.28	8.25	0.34	7.67	0.34	6.58	0.35

Table 4.9: Average number of iterations, and average value of $|N_S^{\texttt{final}}|/|N_T|$.

Refinement trends

We observe the same trends as in Section 4.4.1 for the refinement step in DDD. As the iterations progress, correcting violated storage constraints increases in its impact on the refinement process, whereas there are fewer violated throughput constraints. Correcting short arcs continues to dominate the reason why most timed arcs are infeasible, while the cause of added timed nodes is split between correcting short timed arcs and correcting exceeded storage arcs.

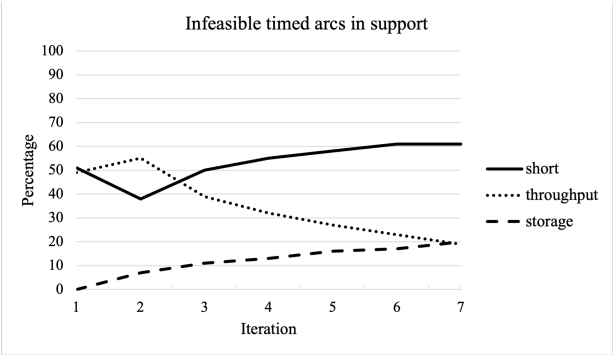


Figure 4.19: Infeasible timed arcs.

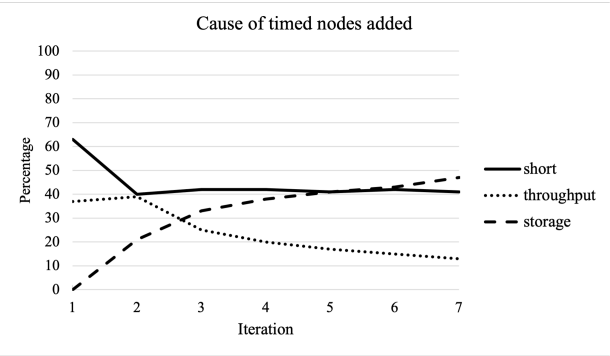


Figure 4.20: Cause of adding timed nodes

4.5 Importance of a tight storage bound

To demonstrate the importance of tightening the storage relaxation in the discrete-time setting we will look at the problem of SND with the addition of hard arc and node capacities. We restate the definition of SND for reference. In a discrete-time instance of SND, we are given a directed graph $D = (N, A)$, where each arc $vw \in A$ has an associated transit time $\tau_{vw} \in \mathbb{N}_{\geq 0}$, a commodity-dependent per-unit-of-flow cost $c^k_{vw} \in \mathbb{R}_{\geq 0}$ for each $k \in \mathcal{K}$, a fixed cost $f_{vw} \in \mathbb{R}_{\geq 0}$, and a capacity $u_{vw} \in \mathbb{N}_{>0}$. In addition, we add hard node and arc capacities. Specifically, we are given a limit of h_{vw} trucks that can be sent along vw at any (integer) point in time, and each node $v \in N$ can store at most b_v units at any time.

Let $\mathcal{K}$ denote a set of commodities, each with an origin $o_k \in N$ and destination $d_k \in N$, along with a demand q_k that must be routed along a single trajectory from s_k to t_k. Let r_k and l_k be the release time and deadline for commodity $k \in \mathcal{K}$ respectively. An o_k, d_k-trajectory is *feasible* for commodity k if it departs o_k no earlier than r_k and arrives at d_k no later than l_k. The goal of SND is to determine a feasible trajectory for each commodity in order to minimize the total fixed and variable cost, ensuring that the hard node and arc capacities are satisfied.

We construct our instance of SND as follows. The base graph, D, is the directed graph depicted in Figure 4.21. Each arc in the figure is also labelled with its transit time and truck capacity. The nodes v and w are labelled with their storage capacity, and the nodes s and t have no storage limit. Let $T \geq 4$.

- $f_a = 1$ and $h_a = 1$ for all $a \in A$;
- $c^k_a = 0$ for all $a \in A$, $k \in \{1, 2\}$;
- $(o_1, d_1) = (o_2, d_2) = (s, t)$;
- $q_k = 50, r_k = 0$ and $l_k = T$ for $k \in \{1, 2\}$.

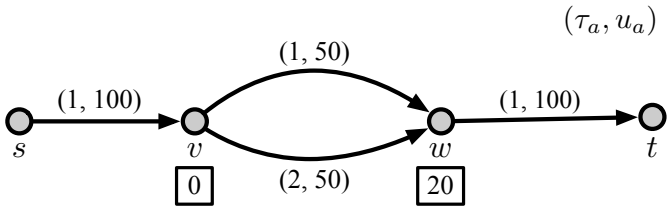


Figure 4.21

Observe that while we would like to purchase only a single truck on each of arcs sv and wt, this is not possible since the two vw arcs have differing transit times and there is

insufficient storage capacity at v and w to hold the units travelling along the quicker route. Thus, the minimum cost of a solution is 5.

Consider an iteration of DDD which includes all timed copies of s, all copies of v, and copies of the node w only at times 0, 1, 2, and T. Note that it is not hard to make this the minimal set of timed nodes permitted in a partial network by adding commodities with restrictive release times and deadlines.

Without the analysis justifying the property ($P^{\texttt{storage}}$), the storage constraint would need to be fully removed at $(w, 2)$ since $(w, 3) \notin N_S$, leading to a solution in the partial network with cost 4. With this weaker relaxation approach, the DDD algorithm would only terminate once $\Omega(T)$ copies of w are added to the solution since every feasible time that w could be reached via commodities 1 and 2 must be included, as well as the following hour to prevent a relaxation of the storage capacity. This would result in $\Omega(T)$ iterations for the DDD algorithm.

Furthermore, without the proof of Lemma 4.5, the combination of Lemmas 4.4 and 4.6 would result in the following weaker (in terms of strength of the relaxation) relaxed capacity property:

($P^{\texttt{storage-relaxed}}$): For any $e = ((v, t), (v, t')) \in H_S$,

$$b'_e \geq \mathtt{m_S}(v, t) \cdot b_v + U_e,$$

where

$$U_e = \sum_{(w,t') \in N_T^-(v,t) \cup N_S^-(v,t)} u_{wv} \cdot (\mathtt{m_S}(w, t') - 1).$$

However, this storage relaxation gives a similarly poor bound on the number of iterations in a DDD algorithm. Since each timed copy of the neighbouring node v is included in N_S, $U_e = 0$ and so we can set the storage capacity of timed node (w, t) to be $\mathtt{m_S}(w, t) \cdot b_w$. Whenever $\mathtt{m_S}(w, t) \geq 3$, the storage will be at least 50, allowing the partial network to permit commodities 1 and 2 to overlap on sv and wt. Thus, the DDD algorithm will only terminate once each copy of w that could be reached via commodities 1 and 2 has $\mathtt{m_S}(w, t) \leq 2$ and so the algorithm would still required $\Omega(T)$ iterations.

With the tighter storage limit given by ($P^{\texttt{storage}}$), the capacity at $(w, 2)$ is only 40, and thus prevents the infeasible solution. As a result, with storage capacities selected according to ($P^{\texttt{storage}}$) the DDD algorithm terminates after a single iteration.

4.6 Two-phase DDD for geographic setting

The two-phase DDD approach was introduced recently by Hewitt [30] as a method to speed-up the initial iterations of DDD. In the first phase, the DDD paradigm solves the LP relaxation of the mixed-integer program. In each iteration in this phase, the LP relaxation of the MIP defined on the partially time-expanded network is solved rather than the MIP itself, in order to save on computation time. The first phase then terminates with an optimal solution to the LP relaxation and a final partially time-expanded network, D_S^{LP}. In the second phase, DDD solves the original MIP with the partially time-expanded network initialized to be D_S^{LP}. This two-phase method was demonstrated to produce optimal solutions more quickly than the single-phase DDD approach for variants of SND [30, 54].

While two-phase DDD is a promising speed-up strategy, one downside of this approach is that the support of an optimal solution to the LP relaxation may be larger than the support of an optimal solution to the MIP, resulting in iterations with larger partially time-expanded networks compared to the partially time-expanded networks of the single-phase approach.

In this section we present computational results comparing a two-phase approach and the original DDD approach on the set of geographic instances presented in Section 4.4.1. For the lower bound in each iteration, we solve the LP relaxation induced by D_S and obtain a solution $\hat{x}$ to UPR(D_S) with value $\hat{T}$. Note that the final arrival time of a timed movement arc in the support of $\hat{x}$, denoted $\texttt{final}(\hat{x})$, could exceed $\hat{T}$ when solving the LP relaxation due to the allowance of fractional variables. Thus, instead of setting $T' = \lceil (1+\delta)\hat{T} \rceil$ as in the original DDD approach, we set $T' = \lceil (1+\delta)\texttt{final}(\hat{x}) \rceil$ in phase one. Furthermore, packets are no longer forced to travel along a single trajectory in the LP relaxation. Let $\hat{\mathcal{Q}}_k = \{\hat{Q}_k^1, \cdots, \hat{Q}_k^{r_k}\}$ be the set of trajectories for packet k in the support of $\hat{x}$, and let $\mathcal{P}_k = \{P_k^1, \cdots, P_k^{r_k}\}$ be the corresponding set of paths in D. For each $k \in \mathcal{K}$ we redefine A_T^k and H_T^k as

$$A_T^k = \{((v,t),(w,t')) \in A_T : vw \in P_k^i \in \mathcal{P}_k\}, \text{ and}$$
$$H_T^k = \{((v,t),(v,t')) \in H_T : v \in N(P_k^i), P_k^i \in \mathcal{P}_k\}.$$

The proof of correctness for the first phase now follows directly from the proof of correctness for the original DDD approach. However, we note that Corollary 4.10 no longer holds for a two-phase approach and instead in the first phase timed nodes (v,t) may be added where $t > T^* + 1$. However, we still observe that for any added timed node (v,t), it must be that $t \leq T$, the provided upper bound. It follows that the maximum number of iterations in phase 1 is at most $|N|T$ (a weakening of Theorem 4.11 and Corollary 4.12).

Overall we found that a two-phase DDD approach was on average *slower* than solving the instance with the original single-phase DDD approach. Specifically, on average the two-phase DDD algorithm was 20.4% slower than the original DDD algorithm. As previously mentioned, there are a few reasons this is not entirely surprising. While the initial iterations can be solved more quickly if we only solve the LP relaxation, these iterations have very sparse partially time-expanded networks, and so naturally the later iterations dominate the runtime of the algorithm. In the following tables, we see that on average the 2-phase approach requires a total of 8.72 iterations, whereas the single-phase DDD approach terminates after an average 7.91 iterations. Additionally, we find an increase of 10% in the average value of $|N_S^{final}|/|N_T|$ (the final timed node set over the full timed node set) when using the two-phase approach.

Results

In Tables 4.10 and 4.11 we compare two-phase and single-phase DDD (original) in terms of their runtime, total number of iterations, and average value of $|N_S^{final}|/|N_T|$.

	average runtime (s)			average $\lvert N_S^{final}\rvert/\lvert N_T\rvert$		
UB	original	2-phase	$\frac{\text{2-phase}}{\text{original}}$	original	2-phase	$\frac{\text{2-phase}}{\text{original}}$
T^*	569	616	1.08	0.58	0.63	1.08
$1.5T^*$	1,019	1,206	1.18	0.39	0.44	1.13
$2T^*$	1,094	1,422	1.30	0.30	0.33	1.10

Table 4.10: Average runtime and average value of $|N_S^{final}|/|N_T|$.

Across all upper bound factors ($T \in \{T^*, 1.5T^*, 2T^*\}$), on average the original DDD approach terminates more quickly than the two-phase approach. One contributing factor is the fact that on average, the size of the final timed node set is larger when running two-phase DDD, resulting in slower final iterations. Additionally, in Table 4.11, we see that the total number of iterations increases when running the two-phase approach. After solving the LP relaxation with DDD, the two-phase approach required an average of approximately 2 iterations in the second phase. Since the later iterations dominate the runtime of DDD and the two-phase approach resulted in slower final iterations, this explains why the two-phase approach does not offer improvement over the original DDD approach for this particular problem and DDD implementation.

	two-phase DDD			original DDD
UB	phase 1	phase 2	total	total
T^*	6.62	2.04	8.65	7.86
$1.5T^*$	6.63	2.27	8.90	7.85
$2T^*$	6.57	2.04	8.60	8.01

Table 4.11: Average number of iterations in each phase.

The following figures present the cumulative instances solved whe $T = 1.5T^*$ and $T = 2T^*$.

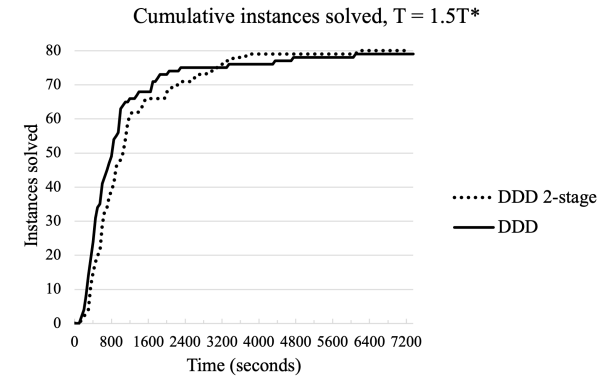


Figure 4.22: $T = 1.5T^*$

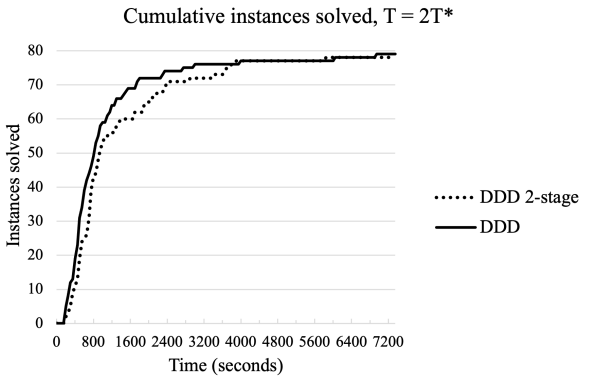


Figure 4.23: $T = 2T^*$

The following tables provide a breakdown of the performance of the two-phase DDD for each value of m, the number of arcs in the base graph. Each entry in the table is equal to the average value of that feature for the two-phase approach divided by the average value of the feature for the original single-phase DDD approach. For example, among the instances where $m = 30, k = 200$, and $T = T^*$, the average number of iterations for two-phase DDD was 11.44 and the average number of iterations for the original DDD approach was 10.56, giving a ratio of $11.44/10.56 = 1.08$.

	$k = 200$			$k = 250$			$k = 300$		
UB	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$
T^*	0.69	1.08	1.06	0.89	1.00	1.03	1.12	1.15	1.38
$1.5T^*$	0.88	1.10	1.05	1.56	1.15	1.27	1.20	1.24	1.14
$2T^*$	1.89	1.02	1.10	1.46	1.09	1.24	1.27	1.19	1.06

Table 4.12: Average runtime when $m = 30$.

	$k = 200$			$k = 250$			$k = 300$		
UB	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$
T^*	1.34	1.12	1.07	1.36	1.14	1.08	1.43	1.19	1.11
$1.5T^*$	1.23	1.07	1.08	1.58	1.17	1.13	1.75	1.20	1.20
$2T^*$	1.01	1.03	1.07	2.03	1.17	1.15	1.32	1.10	1.10

Table 4.13: Average runtime when $m = 45$.

	$k = 200$			$k = 250$			$k = 300$		
UB	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$	time	iterations	$\frac{\lvert N_S^{final}\rvert}{\lvert N_T\rvert}$
T^*	1.08	1.02	0.99	1.51	1.20	1.10	1.08	1.05	1.00
$1.5T^*$	1.14	1.06	1.12	1.52	1.15	1.12	0.66	1.05	1.11
$2T^*$	1.16	0.98	1.08	1.15	1.02	1.06	0.90	1.05	1.12

Table 4.14: Average runtime when $m = 60$.

Consistently we see that the average number of iterations and the average value of $|N_S^{final}|/|N_T|$ increases for the two-phase DDD approach. The same can be said for the number of iterations.

Note that the two-phase approach is not necessarily well-defined even if the DDD algorithm is well-defined for a given MIP. Specifically, without the addition of constraint (4.8), adding timed nodes to D_S could cause the value of the LP relaxation to *decrease* for UPR. Furthermore, it could be the case that the refinement procedure impacts the effectiveness of a two-phase approach, since an aggressive refinement process that corrects all short arcs in the support of a solution may add more timed nodes when given a fractional solution with larger support. Therefore it would be interesting to understand how the formulation and the refinement process can impact the effectiveness of a two-phase DDD approach.

4.7 Summary and observations

In this work, we study the universal packet routing problem and develop a novel DDD algorithm for solving this problem exactly. We prove bounds on the relaxation required for storage constraints for UPR, which extend to the setting of SND with node storage. Moreover, our lower bound arguments rely solely on the structure of the standard map μ from the fully time-expanded network to the partially time-expanded network that is frequently used in the DDD literature. For this reason, the results apply more broadly to any problem with bounded fixed bounded node storage.

We present an implementation along with illustrative computational results. In both our geographic and geometric instances, we observe that the runtime of the full IP increases more than the DDD algorithm as a function of the upper bound, T, provided. In many problems to which DDD has been previously applied, the support of an optimal solution is small compared to the fully time-expanded network. In UPR, this is no longer the case since the arc capacity and storage levels increase the size of the support of a solution.

Finally, we observe that the performance of the full IP deteriorates as the upper bound provided increases relative to T^*. While continuous formulations are commonly said to perform more poorly than time-indexed formulations [39], this will likely no longer be the case for some value of upper bound $T \gg T^*$. Furthermore, it is possible that continuous formulations are more effective when paths are given in advance, such as is the case for the UPR-FP. Potential future work would be to study the comparative runtime between continuous formulations and time-expanded formulations for problems when we add designated paths in the underlying static graph. Understanding this trade-off between continuous and time-indexed formulations may be of interest and could be used to improve upper bound and augmentation steps in DDD algorithms.

Acknowledgements

We thank Cristiana L. Lara for valuable discussions and feedback.

Chapter 5

Additional contributions to DDD

In this chapter we present additional contributions to the development of DDD algorithms. We present a DDD model for solving a version of SND with cyclic constraints, often used to ensure operations are consistent and repeat each day. We show that these cyclic constraints lead to larger sets of timed nodes required in partial networks.

We also address the proposed strategy of reducing formulation sizes by removing timed nodes from the current partial network if they are no longer required in an optimal solution. We show that such a removal strategy must be designed carefully, since straightforward strategies can result in the iterations in DDD to repeat, preventing termination. These results highlight the importance of the arc-based DDD approach presented in Chapter 3.

5.1 Modelling cyclic constraints with DDD

In many practical networks, commodity demands and network operations often have a repetitive structure. That is, the origin-destination flow and corresponding trajectories of packages released on day one are roughly repeated each subsequent day. In addition, the operation of the network is also often repeated each day. For example, often a similar number of trucks are installed on a fixed arc at the same time each day. While in reality day-to-day operations are not exactly the same, such an assumption is at times made in the planning stage of network design [6, 40]. With these assumptions, the problem of computing the optimal operation of a network simplifies to finding an optimal *daily* operation, while taking into consideration the flow released over the course of the whole time horizon.

To compute an optimal daily network operation while incorporating all flow throughout the time horizon, Lara et al. [40] incorporate *cyclic capacity constraints*. For each arc $a \in A$ and hour $t \in [24]$, a cyclic constraint for arc a at time t combines flows departing arc a at time t during *any* day. In the implementation of this constraint, we use the modulo operation. We present these cyclic constraints explicitly in Section 5.1.1. In this section we present a DDD algorithm for SND with cyclic constraints.

5.1.1 SND with cyclic constraints

First, recall the inputs to an instance of SND defined in Section 2. The input consists of the flat network $D = (N, A)$ and commodity set $\mathcal{K}$. Each arc $vw \in A$ has a transit time τ_{vw}, commodity-dependent per-unit-of-flow cost $c^k_{vw} \in \mathbb{R}_{>0}$ for each $k \in \mathcal{K}$, a fixed cost $f_{vw} \in \mathbb{R}_{>0}$, and a capacity $u_{vw} \in \mathbb{N}_{>0}$. Each commodity $k \in \mathcal{K}$ has a source $o_k \in N$, sink $d_k \in N$, demand q_k, release time r_k, and deadline l_k. $T = \max_{k \in \mathcal{K}} l_k$. The goal of SND is to determine a feasible trajectory in D_T for each commodity in order to minimize the total fixed and variable cost of the resources required to ship the commodities along the trajectories within their available time windows.

In an instance of SND with cyclic constraints, we are additionally given a discretization, M, of a day, and T is some multiple of M. The task is to find a min cost feasible routing of the commodities, subject to cyclic constraints.

For each timed arc $a = ((v, t), (w, t')) \in A_T$, let $\texttt{dep}(a)$ denote the departure time of a. That is, $\texttt{dep}(a) = t$. Let $A(a) := vw$. i.e., $A(\cdot)$ projects timed arcs down to the base graph.

To precisely state the constraints, we adopt the same notation as in Section 2. For each commodity $k \in \mathcal{K}$ and each timed arc $a \in A^k_T \cup H^k_T$, let x^k_a be a binary variable which is equal to 1 if commodity k is scheduled to travel along timed arc a in its assigned trajectory. For each $a \in A_T$, let y_a denote the number of trucks scheduled along timed arc a.

The capacity constraints for traditional SND were as follows.

$$\sum_{k \in \mathcal{K}} q_k x^k_a \leq u_a y_a \quad \forall a \in A_T$$

Let y^t_{vw} denote the total number of trucks schedules on arc vw, at any time $\bar{t}$ where $\bar{t} \equiv t \pmod M$. When capacity constraints are cyclic, we instead have the following (often smaller) set of constraints.

$$\sum_{\substack{k \in \mathcal{K}, a \in A_T : A(a) = vw, \\ \texttt{dep}(a) \equiv t \pmod M}} q_k x^k_a \leq u_{vw} y^t_{vw} \quad \forall vw \in A, t \in [M]$$

The full time-indexed formulation is restated as follows. For each timed arc $a = ((v,t),(w,t')) \in A_T$, let $u_a := u_{vw}$, and $c_a := c_{uv}$.

$$\min \quad \sum_{vw \in A} \sum_{t \in [M]} f_{vw} y^t_{vw} + \sum_{a \in A_T} \sum_{k \in \mathcal{K}} c^k_a q_k x^k_a \qquad \text{(SND-cyclic}(D_T)\text{)}$$

$$\text{s.t.} \quad x^k(\delta^+_{D_T}(v,t)) - x^k(\delta^-_{D_T}(v,t)) = \begin{cases} 1 & (v,t) = (o_k, r_k) \\ -1 & (v,t) = (d_k, l_k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_T \quad (5.1)$$

$$\sum_{\substack{k \in \mathcal{K}, a \in A_T : A(a) = vw, \\ \texttt{dep}(a) \equiv t \pmod M}} q_k x^k_a \leq u_a y^t_{vw} \quad \forall vw \in A, t \in [M] \qquad (5.2)$$

$$x^k_a \in \{0,1\} \quad \forall k \in \mathcal{K}, a \in A_T \cup H_T \qquad (5.3)$$

$$y^t_{vw} \in \mathbb{N}_{\geq 0} \quad \forall vw \in A, t \in [M] \qquad (5.4)$$

We will present a lower bound, upper bound, and refinement procedure for solving the problem of SND with cyclic constraints with DDD. The key challenge is in accommodating the relaxation for the cyclic constraints in the lower bound and refinement steps.

5.1.2 Lower bound model

The following formulation is analogous to the lower bound formulation presented in Section 2.2 for SND. For each timed arc $a = ((v,t),(w,t')) \in A_S$, $\texttt{dep}(a) = t$ and $A(a) = vw$.

$$\min \quad \sum_{vw \in A} \sum_{t \in [M]} f_{vw} y^t_{vw} + \sum_{a \in A_S} \sum_{k \in \mathcal{K}} c^k_a q_k x^k_a \qquad \text{(SND-cyclic}(D_S)\text{)}$$

$$\text{s.t.} \quad x^k(\delta^+_{D_S}(v,t)) - x^k(\delta^-_{D_S}(v,t)) = \begin{cases} 1 & (v,t) = (o_k, r_k) \\ -1 & (v,t) = (d_k, l_k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, (v,t) \in N_S \quad (5.5)$$

$$\sum_{\substack{k \in \mathcal{K}, a \in A_S : A(a) = vw, \\ \texttt{dep}(a) \equiv t \pmod M}} q_k x^k_a \leq u_{vw} y^t_{vw} \quad \forall vw \in A, t \in [M] \qquad (5.6)$$

$$\sum_{a \in A_S} \tau_a x^k_a \leq l_k - r_k, \quad \forall k \in \mathcal{K}. \qquad (5.7)$$

$$x^k_a \in \{0,1\} \quad \forall k \in \mathcal{K}, a \in A_S \cup H_S \qquad (5.8)$$

$$y^t_{vw} \in \mathbb{N}_{\geq 0} \quad \forall vw \in A, t \in [M] \qquad (5.9)$$

Recall that in the case of SND, SND-RR(D_S) provides a lower bound on the optimal value of SND-RR(D_T), so long as the partial network $D_S = (N_S, A_S)$ satisfies the following two properties.

(P1) **Timed nodes**: For all $k \in \mathcal{K}$, (o_k, r_k) and (d_k, l_k) are in N_S;
For all $v \in N$, $(v, 0)$, and (v, T) are in N_S;

(P2) **Arc copies**: For all $vw \in A$, for all $(v, t) \in N_S$ with $t + \tau_{vw} \leq T$, we have $((v, t), (w, t')) \in A_S$ where

$$t' = \max\{r : r \leq t + \tau_{vw}, (w, r) \in N_S\}.$$

However, in the case of SND with cyclic constraints, these properties are not sufficient to guarantee a lower bound. Consider the following instance with base graph D as given in Figure 5.1 with unit transit times. The other inputs are listed below.

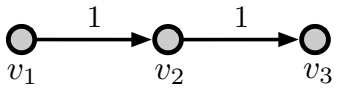


Figure 5.1

Transit times and capacities:

- $\tau_a = f_a = 1$, $u_a = 2$
- $c_a^k = 0 \ \forall a \in A, \forall k \in \mathcal{K}$;
- $M = 2$ and $T = 4$.

Commodities:

- $k_1 : (o_1, d_1) = (v_1, v_3), r_1 = 0, l_1 = 4$;
- $k_2 : (o_2, d_2) = (v_2, v_3), r_2 = 0, l_2 = 1$;
- $q_1 = q_2 = 1$.

Observe that since $M = 2$ and $T = 4$, we can think of the time horizon as two days, and the time discretization as half-day increments. With this interpretation, each arc has a transit time of 12 hours (one half day).

Figure 5.2 provides the full network, and in Figure 5.3 the dashed arcs form a feasible solution for the instance. Observe that the cost of this solution is 2, since the two timed arcs $((v_2, 0), (v_3, 1))$ and $((v_2, 2), (v_3, 3))$ contribute to the same capacity constraint since $1 \equiv 3 \pmod 2$. Note that the support is not necessarily cyclic since the model only includes each commodity once, rather than each day. The cyclic behaviour is captured by the capacity constraints rather than repeating the commodities each day in the model. The partial network presented in Figure 5.4 satisfies both property $(P1)$ and property $(P2)$. However, since the deadline for commodity 2 is time 1, the only feasible routing through D_S has cost 3. Note that even though D_S had *no short timed arcs*, routing through partial network does not guarantee a lower bound when cyclic constraints are present.

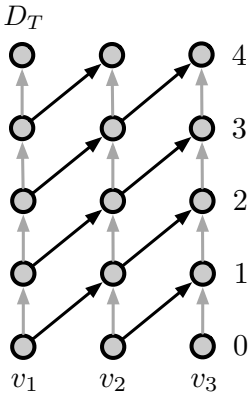


Figure 5.2:
Full network D_T.

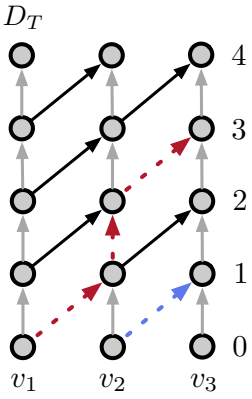


Figure 5.3:
Solution on D_T.

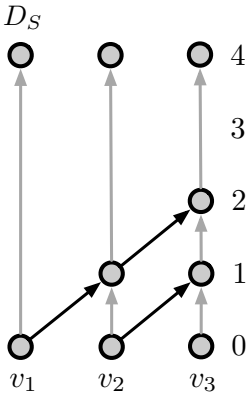


Figure 5.4:
Partial network D_S.

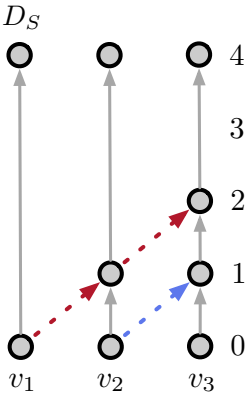


Figure 5.5:
Solution on D_S.

We will prove that adding the following property is sufficient to ensure that routing through D_S provides a lower bound on the cost of routing in D_T. Specifically, we show that it if the time points included for each node with non-zero out-degree are closed under the modular operation in D_S, then the optimal value of SND-cyclic(D_S) provides a lower bound on the optimal value of SND-cyclic(D_T). We refer to a node v as *non-terminal* if $\deg^+_D(v) > 0$.

(P3) **Cyclicity**: For all $v \in N$ with $\deg^+_D(v) > 0$, if $(v, t) \in N_S$ then $(v, t') \in N_S$ for all $t' \in [T]$ with $t' \equiv t \pmod{M}$.

The partial networks in Figures 5.6 and 5.8 each satisfy property (P3). The corresponding optimal routings with cost 2 are given in Figures 5.7 and 5.9.

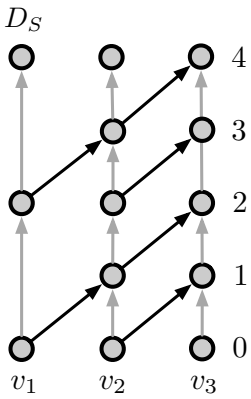


Figure 5.6:
Partial network D_S.

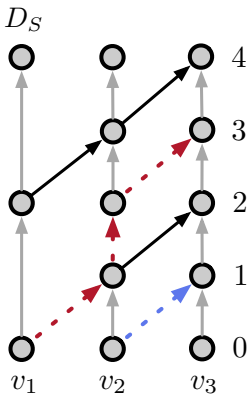


Figure 5.7:
Solution on D_S.

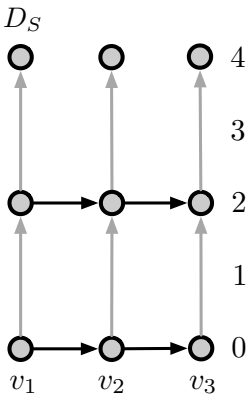


Figure 5.8:
Partial network D_S.

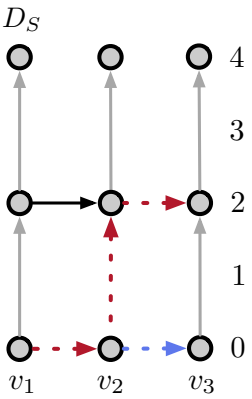


Figure 5.9:
Solution on D_S.

In the proof of the following theorem, we use the standard map μ from timed arcs in A_T to timed arcs in A_S. The only change in the proof compared to the proof of Theorem 2.2 is the consideration of the cyclic constraint and the corresponding cost.

Theorem 5.1. *Let $\mathcal{I}$ be an instance of SND-cyclic. Let $D_S = (N_S, A_S \cup H_S)$ be a partial network that satisfies properties $(P1)-(P3)$. Then an optimal solution to SND-cyclic(D_S) provides a lower bound on the optimal value of SND-cyclic(D_T).*

Proof. Let $\mu : A_T \to A_S$ be the map defined so that for each timed arc $a \in A_T$,

$$a = ((v,t),(w,t')) \quad \to \quad \mu(a) = ((v,\hat{t}),(w,\hat{t}')), \tag{5.10}$$

where $\hat{t} = \max\{s : s \le t, (v,s) \in N_S\}$, and $\hat{t}' = \max\{s : s \le \hat{t} + \tau_{vw}, (v,s) \in N_S\}$. Again, μ is well-defined due to $(P1)$ and $(P2)$.

Let $(\bar{x}, \bar{y})$ be a feasible solution to SND-cyclic(D_T) with cost C, and let $\mathcal{Q} = \{Q_k\}_{k\in\mathcal{K}}$ denote the corresponding set of trajectories. Fix $k \in \mathcal{K}$. The set of ordered movement arcs in Q_k is $\{a_1, a_2, \ldots, a_{q_k}\}$, where each $a_i = ((v_i, t_i^{out}), (v_{i+1}, t_{i+1}^{in}))$ for each $i \in [q_k - 1]$. As in the proof of Theorem 2.2, we obtain feasible trajectories $\hat{\mathcal{Q}} = \{\hat{Q}_k\}_{k\in\mathcal{K}}$ in D_S by mapping each movement arc $a \in A_T$ to $\mu(a) \in A_S$, and forming trajectories by adding holdover arcs. Note that the underlying paths in the flat network for $\hat{\mathcal{Q}}$ and $\mathcal{Q}$ are the same for each commodity, so the variable costs are the same.

It remains to argue that the fixed costs for $\hat{\mathcal{Q}}$ are no larger than those of $\mathcal{Q}$. Let k and j be two commodities that contribute to the same capacity constraint corresponding to arc $vw \in A$ and time $t \in [M]$. That is, k traverses timed arc $a_1 = ((v,t_1),(w,t_1'))$ and j traverses timed arc $a_2 = ((v,t_2),(w,t_2'))$, where $t \equiv t_1 \equiv t_2 \pmod M$. The function μ maps a_1 and a_2 to timed arcs $\mu(a_1) = ((v,\hat{t}_1),(w,\hat{t}_1'))$ and $\mu(a_2) = ((v,\hat{t}_2),(w,\hat{t}_2'))$, where $\hat{t}_1 \equiv \hat{t}_2 \pmod M$ by property $(P3)$. Therefore commodities k and j continue to contribute to the same capacity constraint for arc vw, potentially with a different time $\hat{t} \in [M]$ where $\hat{t} \equiv \hat{t}_1 \pmod M$. Therefore the fixed cost incurred when routing $\hat{\mathcal{Q}}$ in D_S for each base arc in A is at most the cost incurred when routing $\mathcal{Q}$ in D_T, and so the cost of $\hat{\mathcal{Q}}$ is at most C. □

5.1.3 Upper bound and refinement considerations

Let $(\hat{x}, \hat{y})$ be an optimal solution to SND-cyclic(D_S) with corresponding trajectories $\mathcal{Q} = \{Q_k\}_{k\in\mathcal{K}}$ in D_S, and paths $\mathcal{P} = \{P_k\}_{k\in\mathcal{K}}$ in D. A feasible solution to SND-cyclic(D_T) with the same flat paths $\mathcal{P}$ is obtained by specifying a departure time for each arc in P_k, for

each commodity $k \in \mathcal{K}$, that satisfies the release time and deadline of k and respects the actual transit times.

In the case of SND, this feasible solution has the same cost as $\mathcal{Q}$ if every pair of commodities dispatched together on an arc vw in $\mathcal{Q}$ still shares a common departure time for arc vw. However, this is not true in the case of SND with cyclic constraints, since two commodities that traverse different timed copies of an arc $vw \in A$ can still contribute to the same capacity constraint if their departure times are congruent. Due to this difference, the continuous formulation presented in Section 2.2 no longer applies. One choice for an upper bound procedure would be to solve a formulation with the additional restriction that each commodity traverses the flat path dictated by $\mathcal{Q}$, analogous to the approach presented in Section 4.3.

If an optimal solution to the upper bound formulation matches the cost of $\mathcal{Q}$, then an optimal solution to SND-cyclic(D_T) has been found. Otherwise, the current partial network contains short timed arcs and must be modified.

Standard update procedures consist of lengthening the earliest-departing short timed arc in each trajectory in $\mathcal{Q}$, or lengthening all short timed arcs. Let $a = ((v,t),(w,t')) \in A_S$ be a timed arc in the support of $\hat{x}$ that is short. Recall, the standard method to correct short arcs in A_S [2]:

$$\text{if } t' < t + \tau_{vw}, \text{ add the timed node } (w, t + \tau_{vw}).$$

In the case of SND with cyclic constraints, the set of timed nodes in the partial network must be updated so that it continues to satisfy property (P3). Thus, for any timed arc chosen to be lengthened, the following update procedure in Algorithm 13 must be completed when lengthening a timed arc where the head node has departing arcs in D.

Algorithm 13: `Augment-`$N_S(e)$

Input: A short timed arc $a = ((v,t_1),(w,t_2))$ with $\deg_D^+(w) > 0$.
for $t \in \{s \in [T] : s \equiv t_1 + \tau_{vw} \pmod M\}$ **do**
    $N_S \leftarrow N_S \cup (w,t)$
return N_S

5.2 Removing timed nodes

A DDD algorithm often performs well when there are not too many iterations until termination and the corresponding formulations are (very) small compared to the size of the

full time-indexed formulation. Since a reasonable indicator of the size of the lower bound formulation solved in a DDD is the size of the partial network, D_S, we are motivated to maintain small partial networks throughout the algorithm.

In traditional DDD approaches, in each iteration the size of the partial networks grows. In order to maintain small partial networks, Scherr et al. [54] propose the idea of removing timed nodes and timed arcs from the current partial network if they are no longer required for obtaining high-quality solutions. While this general strategy has been proposed, there is little understanding on how it should be implemented, and whether or not it would be successful.

An initial strategy for removing timed nodes would be to remove any timed node that is not used in the current solution to the lower bound model defined on the partial network, so long as properties of the lower bound model can still be satisfied with the remaining timed nodes. However, we prove that this strategy can lead to cycling in DDD.

To demonstrate the potential for this strategy to create cycling, we examine a simple SND instance where the base graph is a cycle. See Section 2 for the definition of SND and Section 2.2 for a standard DDD algorithm for SND.

Recall that a partial network $D_S = (N_S, A_S)$ corresponds to a valid lower bound model so long as properties $(P1)$ and $(P2)$ are satisfied. In terms of the timed node set, N_S, the only requirement is that $(o_k, r_k), (d_k, l_k) \in N_S$ for all $k \in \mathcal{K}$. That is, each node v is included at any time point at which a commodity with origin v has its release time, or a commodity with destination v has its deadline. In the refinement step for the SND model, short timed arcs are lengthened if the solution to the partial network cannot be mapped to a solution to the full network.

We construct an instance of SND as follows. The base graph, D, is the directed triangle depicted in Figure 5.10.

Transit times and capacities:

- $\tau_a = f_a = 1$, $u_a = 2$ for all $a \in A$;
- $c_a^k = 0$ for all $a \in A$, for all $k \in \mathcal{K}$.

Commodities:

- $(o_1, d_1) = (v_1, v_3)$;
- $(o_2, d_2) = (v_2, v_1)$;
- $(o_3, d_3) = (v_3, v_2)$;
- $q_k = 1, r_k = 0$ and $l_k = 4$ for all $k \in \mathcal{K}$.

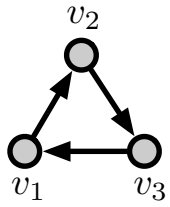


Figure 5.10: Base graph D.

Observe that there is a unique path in the base graph for each commodity. Any pair of commodities can be scheduled to travel along their shared arc at the same time, incurring only the fixed cost of that arc once. However, at least one arc in D must be traversed by the commodity set at more than one time. Thus, the minimum cost is 4.

We now consider the execution DDD algorithm if we were to remove all timed nodes that are not in the current solution in each iteration, provided that $(P1)$ remains satisfied. In this case, we simply need to ensure $(o_k, 0)$ and $(d_k, 4)$ remain in D_S for each $k \in \mathcal{K}$. In Figure 5.11 we present the partial network in each iteration of the DDD algorithm, along with an optimal solution to the lower bound formulation marked in the dashed timed arcs. In each iteration, the lower bound value is 3, and the refinement process lengthens all short timed arcs. When we remove timed nodes that are not in the current optimal solution, we see that iterations 2 and 5 are identical. Thus, removing unused timed nodes causes the DDD iterations to cycle.

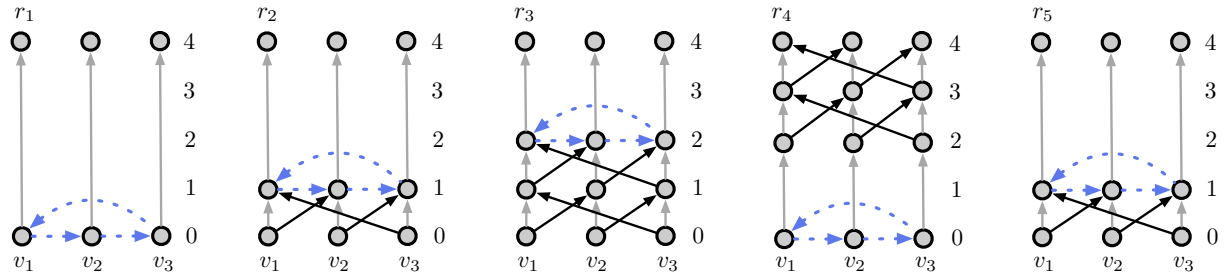


Figure 5.11: DDD algorithm with node removal.

Even in the case where a refinement process only corrects a single timed arc, the same cycling can occur. Figure 5.12 demonstrates that iteration $i > 1$ in Figure 5.11 will be repeated in iteration $3i - 2$ of the DDD algorithm that corrects a single timed arc in each refinement step.

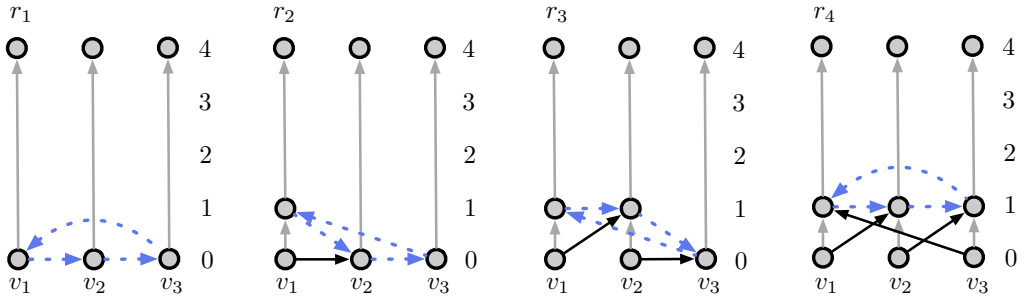


Figure 5.12: DDD algorithm with node removal, with refinement step correcting a single arc.

Thus, while a timed node removal strategy may be helpful, the procedure must be chosen carefully in order to avoid cycling.

5.3 Summary and observations

In this chapter we presented examples demonstrating that an aggressive timed node removal strategy can cause iterations to repeat, preventing termination. We also introduced a DDD approach for the problem of SND with cyclic constraints. Most notably, we proved that so long as the time the discretization for each non-terminal node is cyclic (the same each day) and satisfies previous standard properties, an optimal routing though the partial network provides a lower bound on the optimal value.

Each of these results highlights the challenge of maintaining small partial networks in the generic DDD paradigm. The former suggests challenges in removing timed nodes, and the latter results in an increased number of timed nodes to maintain a valid lower bound formulation. Future work could be to implement a DDD algorithm for SND with cyclic constraints and consider heuristic timed node removal strategies. Another interesting consideration would be to find a continuous formulation for the upper bound step of SND with cyclic constraints.

Part II

Sortation

Chapter 6

Minimum degree sort point problem

In this section we consider the problem of `min-degree-SPP`, and we prove that it is NP-hard to determine if there is a feasible allocation of sort points given a degree bound. We discuss several settings, where (near-)feasibility of a given sortation instance can be determined efficiently. The algorithms we propose are fast and build on combinatorial *witness set* type lower bounds that are reminiscent and extend those used in earlier work on degree-bounded spanning trees and arborescences.

See Section 1.4 for the definition of `min-degree-SPP` and relevant notation.

6.1 Hardness

In this section we prove that `min-degree-SPP` is NP-hard, even in the setting where the underlying undirected graph of D forms a star. To prove this result, we will exhibit a reduction from the NP-hard problem of `Hitting-set` [23], defined as follows.

`Hitting-set`: Let $\Sigma = \{e_1, e_2, \ldots, e_m\}$ be a set of m elements, let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be a set of n non-empty subsets of Σ, and let $b \in \mathbb{N}$ be a budget. The hitting set problem asks if there is a subset $R \subset \Sigma$ of cardinality at most b such that $R \cap S_i \neq \emptyset$ for all $i \in [n]$.

Theorem 6.1. `min-degree-SPP` *is NP-hard, even when restricted to star instances.*

Proof. Let $\mathcal{H} = (\Sigma, \mathcal{S}, b)$ be an instance of `Hitting-set`, where $\Sigma = \{e_1, e_2, \ldots, e_m\}$, $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ is a set of n non-empty subsets of Σ, and $b \in \mathbb{N}$. The hitting set problem asks if there is a subset $R \subset \Sigma$ of cardinality at most b such that $R \cap S_i \neq \emptyset$ for

all $i \in [n]$. We will construct a corresponding instance $\mathcal{I} = (D, \mathcal{K})$ of `min-degree-SPP` such that $\mathcal{H}$ is a `YES` instance if and only if $\Delta^* \leq c$, for some fixed integer c, where $\mathcal{I}$ and c are polynomial in the size of $\mathcal{H}$.

First, we construct a digraph $D' = (N', A')$ with node and arc sets

$$N' = \{s_i : i \in [n]\} \cup \{v\} \cup \{t_j : j \in [m]\},$$
$$A' = \{s_i v : i \in [n]\} \cup \{v t_j : j \in [m]\},$$

as shown in Figure 6.1a. For each $i \in [n]$ and $e_j \in S_i$, we add a commodity with source s_i and sink t_j to form a set $\mathcal{K}'$. That is, $\mathcal{K}' = \{(s_i, t_j) : i \in [n], e_j \in S_i\}$.

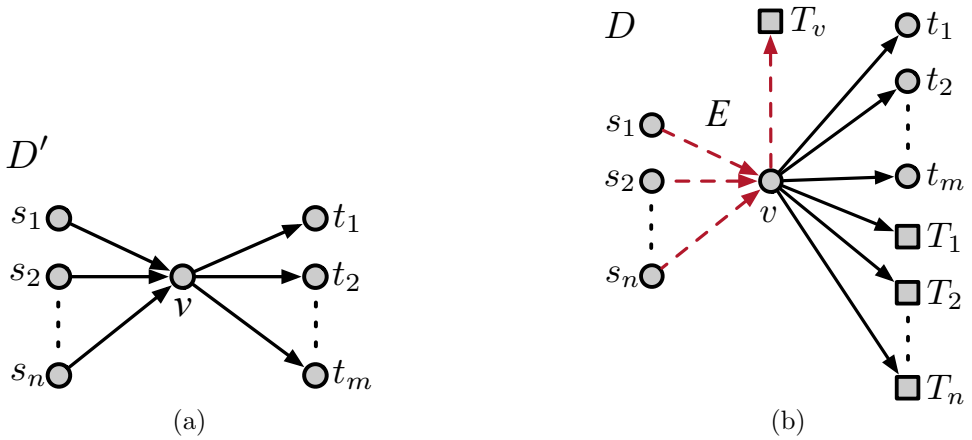


Figure 6.1: Digraphs D' and D.

We build on the instance $\mathcal{I}' = (D', \mathcal{K}')$ to form instance $\mathcal{I} = (D, \mathcal{K})$. Let $c = \max\{b, \max_i |S_i|\}$. For each $i \in [n]$, we add a set of nodes, denoted T_i, of cardinality $c - |S_i|$ to N'. Additionally, we add a set of nodes T_v with cardinality $c - b$. The arc set A is formed by adding an arc from v to all the nodes in $N \setminus N'$. That is,

$$N = N' \cup T_v \cup \{T_i : i \in [n]\}$$
$$A = A' \cup \{vt : t \in N \setminus N'\}.$$

The digraph D is given in Figure 6.1b, where the square nodes indicate a set of nodes rather than a single node. Additionally, arcs entering a square node denote a set of arcs with same shared tail and differing heads – one for each node in the set represented by the square node.

We add commodities to $\mathcal{K}'$ to form the set $\mathcal{K}$. For each $i \in [n]$, we add the commodity (s_i, t) for all $t \in T_i$, and the commodity (s_i, v). Additionally, for each node $t \in T_v$ we define

a commodity (v, t). Specifically,

$$\mathcal{K} = \mathcal{K}' \cup \{(s_i, v) : i \in [n]\} \cup \{(s_i, t) : t \in T_i, i \in [n]\} \cup \{(v, t) : t \in T_v\}.$$

Observe that $\mathcal{I}$ and c are polynomial in the input of $\mathcal{H}$. Furthermore, there are $c + 1$ distinct commodities with source s_i for each $i \in [n]$, and any feasible subgraph H must contain the arc set $E = \{s_i v : i \in [n]\} \cup \{vt : t \in T_v\}$ (the red dashed arcs in Figure 6.1b). We now claim that the hitting set instance $\mathcal{H}$ is a `YES` instance if and only if $\Delta^* \leq c$.

First, suppose the hitting set instance $\mathcal{H}$ is a `YES` instance. Let R be a set of at most b elements such that $R \cap S_i \neq \emptyset$ for all $i \in [n]$. Up to reordering, we may assume that $R = \{e_1, e_2, \ldots, e_p\}$ where $p \leq b$. We form a feasible solution H as follows. First, let H_1 be the subgraph containing the edge set E along with the set $\{vt_i : i \in [p]\}$, as shown in Figure 6.2. We define the digraph H_2 as the subgraph of $\texttt{cl}(D)$ with arc set $\{s_i t : t \in T_i, i \in [n]\} \cup \{s_i t_j : e_j \in S_i \setminus R, i \in [n]\}$. See Figure 6.2 for an example of H_2.

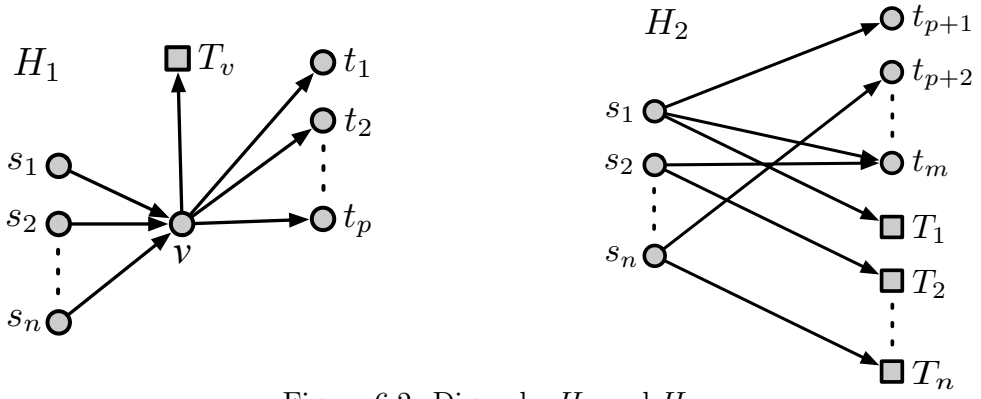


Figure 6.2: Digraphs H_1 and H_2.

Let $H = H_1 \cup H_2$. We claim that H is feasible and $\Delta^+(H) \leq c$. Let $k \in \mathcal{K}$. If $s_k = v$, then $t_k \in T_v$ and H_1 contains an s_k, t_k-dipath. Otherwise, $s_k = s_i$ for some $i \in [n]$. If $t_k = t_j$ for some $e_j \in S_i \cap R$ or $t_k = v$, then similarly, H_1 contains an s_k, t_k-dipath. The final case is if $t_k = t_j$ for some $e_j \in S_i \setminus R$ or $t_k \in T_i$, in which case H_2 contains an s_k, t_k-dipath. Thus, H is feasible.

It remains to argue that $\Delta^+(H) \leq c$. For each $i \in [n]$,

$$\deg^+_H(s_i) = 1 + \deg^+_{H_2}(s_i) = 1 + |T_i| + |S_i \setminus R| = 1 + c - |S_i| + |S_i \setminus R| \leq c,$$

where the inequality follows since $R \cap S_i \neq \emptyset$ for all $i \in [n]$. Additionally, we have

$$\deg^+_H(v) = p + |T_v| \leq b + (c - b) = c.$$

We now prove the reverse direction. Suppose $H \subseteq \texttt{cl}(D)$ is a feasible subgraph for instance $\mathcal{I}$, with $\Delta^+(H) \leq c$. We may assume that H is minimal in the sense that removing any arc results in an infeasible solution. This minimality ensures that for each $i \in [n]$, the heads of arcs with tail s_i in H must be in the set $v \cup T_i \cup \{t_j : e_j \in S_i\}$, which has cardinality $c+1$.

Let B be the set of arcs in H from v to some node in $\{T_i : i \in [n]\}$. We will argue that we may assume this set is empty. Suppose $B \neq \emptyset$. That is, H contains an arc vt for some $t \in T_i$ for some $i \in [n]$. By minimality, $H - vt$ is infeasible, and so H does not contain the arc $s_i t$. If $\deg^+_H(s_i) < c$, then $H - vt + s_i t$ is a minimal feasible solution with one fewer node in B. Otherwise, $\deg^+_H(s_i) = c$. This implies that H contains an arc st_j for some $e_j \in S_i$, since the set S_i is non-empty. Then $H' = H - vt - s_i t_j + s_i t + vt_j$ is a minimal feasible solution with one fewer arc in B. By repeating this argument, we see that we may assume $B = \emptyset$.

Since H must contain the arc set E, and $\deg^+_E(v) = |T_v| = c - b$, H contains at most b arcs from the set $\{vt_j : j \in [m]\}$. Furthermore, for each $i \in [n]$ there are $c+1$ commodities (each with a distinct sink) which have s_i as the source. Since $\deg_H(s_i) \leq c$, at least one of these commodities must be routed by a path of length two. That is, for each s_i there is some commodity with source s_i and sink t where vt is an arc in H. Since $B = \emptyset$, H must contain an arc vt_j for some $e_j \in S_i$. Therefore, $R = \{e_j : vt_j \in H, j \in [m]\}$ forms a hitting set of $\mathcal{S}$ of cardinality at most b. □

6.2 Combinatorial lower bounds

In this section we present a family of lower bounds, defined by a set of nodes W and set of commodities $\mathcal{K}'$, which we refer to as a *witness set*.

Let $D = (N, A)$ be a digraph, and consider a subset $U \subseteq N$. We define $\delta^+_D(U)$ as the set of arcs in D leaving U. That is, $\delta^+_D(U) := \{vw \in E : v \in U, w \notin U\}$.

Consider a subset $W \subseteq N$ such that $s_k \in W$ and $t_k \notin W$ for some $k \in \mathcal{K}$. It follows that any feasible subgraph H must contain some arc in $\delta^+_{\texttt{cl}(D)}(W)$. Specifically, due to the given-path structure, we know that H must contain some arc in $\delta^+_{\texttt{cl}(P_k)}(W)$. In Figure 6.3, consider the node set $W = \{s, v\}$ along with the commodity with source s and sink t_1. On the left is the base graph D, and on the right is the transitive closure. Since $s \in W$ and $t_1 \notin W$, it follows that any feasible solution H must have a non-empty intersection with the set $\{st_1, vt_1\}$.

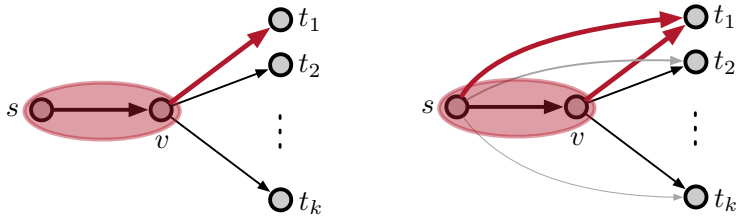


Figure 6.3: A tree instance with k commodities $\{(s, t_i) : i \in [k]\}$.

We combine disjoint cuts for a fixed node set W to obtain lower bounds on the value of Δ^*. To simplify the set of cuts considered, we prove the following lemma to work with cuts in D rather than in its closure.

Lemma 6.2. *Let $\mathcal{I} = (D, \mathcal{K})$ be a tree instance and $W \subseteq N$. For any $k, j \in \mathcal{K}$,*

$$\delta^+_{P_k}(W) \cap \delta^+_{P_j}(W) = \emptyset \text{ if and only if } \delta^+_{\texttt{cl}(P_k)}(W) \cap \delta^+_{\texttt{cl}(P_j)}(W) = \emptyset.$$

Proof. Suppose $\delta^+_{P_k}(W) \cap \delta^+_{P_j}(W) = \emptyset$ and for a contradiction, suppose there is an arc $vw \in \delta^+_{\texttt{cl}(P_k)}(W) \cap \delta^+_{\texttt{cl}(P_j)}(W)$. Then $v \in W$, $w \notin W$, and both P_k and P_j contain a v, w-dipath. Since the underlying undirected graph of D is a tree, it follows that P_k and P_j contain the *unique* v, w-dipath in D. Furthermore, since $v \in W$ and $w \notin W$, there is some arc xy on the v, w-dipath where $x \in W$ and $y \notin W$. As a result, $xy \in \delta^+_{P_k}(W) \cap \delta^+_{P_j}(W)$, a contradiction.

The reverse direction immediately follows from the observation that $\delta^+_F \subseteq \delta^+_{\texttt{cl}(F)}$ for any directed graph F. Thus, $\delta^+_{P_k}(W) \cap \delta^+_{P_j}(W) \subseteq \delta^+_{\texttt{cl}(P_k)}(W) \cap \delta^+_{\texttt{cl}(P_j)}(W)$. □

As is standard in proving bounds on the min-max degree [21, 65], we observe the following: if ℓ distinct arcs must leave a set W of nodes in any feasible solution, then $\Delta^* \geq \lceil \ell / |W| \rceil$. The following lower bound construction shows that we can argue that such a disjoint arc set can be derived by looking at disjoint cuts of the form $\delta^+_{P_k}(W)$ for some $k \in \mathcal{K}$. Further, we show that this lower bound can be strengthened since we must also have connectivity *within* W in order to allocate the arcs departing W to different nodes in W.

Lemma 6.3. *Let $\mathcal{I} = (D, \mathcal{K})$ be a tree instance of* `min-degree-SPP`*. Let $W \subseteq N$ such that $D[W]$ is weakly-connected and suppose $\emptyset \neq \mathcal{K}' \subseteq \mathcal{K}$ such that $s_k \in W$ and $t_k \notin W$ for all $k \in \mathcal{K}'$, and $\delta^+_{P_k}(W) \cap \delta^+_{P_j}(W) = \emptyset$ for all distinct $k, j \in \mathcal{K}'$. Then*

$$\Delta^* \geq \left\lceil \frac{|\mathcal{K}'| + |W| - |\mathcal{S}(\mathcal{K}')|}{|W|} \right\rceil,$$

where $\mathcal{S}(\mathcal{K}')$ denotes the set of sources for commodities in $\mathcal{K}'$.

Proof. Let F be a feasible subgraph of $\texttt{cl}(D)$, and let $W, \mathcal{K}'$ be given as in the statement. Let H be the subgraph of F where arcs are removed from F if the remaining subgraph remains feasible for the commodity set $\mathcal{K}'$. That is, H is a minimal subgraph of F that contains an s_k, t_k-dipath for each commodity $k \in \mathcal{K}'$.

Since $s_k \in W$ and $t_k \notin W$, for each $k \in \mathcal{K}'$ it follows $\delta^+_{\texttt{cl}(P_k)}(W) \cap H \neq \emptyset$. Since $\delta^+_{P_k}(W) \cap \delta^+_{P_j}(W) = \emptyset$ for all distinct $k, j \in \mathcal{K}'$, Lemma 6.2 implies that for all distinct $k, j \in \mathcal{K}'$, $\delta^+_{\texttt{cl}(P_k)}(W) \cap \delta^+_{\texttt{cl}(P_j)}(W) = \emptyset$. Thus, any feasible subgraph must have at least $|\mathcal{K}'|$ arcs in $\delta^+_{\texttt{cl}(D)}(W)$, and so

$$\Delta^* \geq \left\lceil \frac{|\mathcal{K}'|}{|W|} \right\rceil.$$

If for any commodity $k \in \mathcal{K}'$ there is no arc $s_k u \in \delta^+_{\texttt{cl}(P_k)}(W) \cap H$, then since H is feasible, there is an arc $v_k u \in \delta^+_{\texttt{cl}(P_k)}(W) \cap H$ along with an s_k, v_k-dipath in H. Since $D[W]$ is weakly-connected and $\mathcal{I}$ is a tree instance, any s_k, v_k-dipath only uses arcs between nodes in W.

Let A' be a set of at least $|\mathcal{K}'|$ arcs in $\delta^+_{\texttt{cl}(D)}(W) \cap H$, where $A' \cap \delta^+_{\texttt{cl}(P_k)}(W) \neq \emptyset$ for all $k \in \mathcal{K}'$. Moreover, select the arcs in A' so that for each $k \in \mathcal{K}'$, either an arc $s_k u \in A'$ or $v_k u \in A'$ where H has an s_k, v_k-dipath. Thus, if there are $\ell \geq 1$ nodes in W with departing arcs in A', H contains at least $\ell - |\mathcal{S}(\mathcal{K}')|$ arcs with both endpoints in W. Therefore,

$$\Delta^* \geq \min_{|\mathcal{S}(\mathcal{K}')| \leq \ell \leq |W|} \left\lceil \frac{|\mathcal{K}'| + \ell - |\mathcal{S}(\mathcal{K}')|}{\ell} \right\rceil = \left\lceil \frac{|\mathcal{K}'| + |W| - |\mathcal{S}(\mathcal{K}')|}{|W|} \right\rceil.$$

This follows from the fact that $f(x) = \frac{\alpha + x}{x}$ is non-increasing for $\alpha \geq 0, x \geq 1$. □

We define the functions $\texttt{LB}$ and $\texttt{LB}^w$, where for each $W \subseteq N$ and $\mathcal{K}' \subseteq \mathcal{K}$,

$$\texttt{LB}_{\mathcal{I}}(W, \mathcal{K}') := \begin{cases} \left\lceil \frac{|\mathcal{K}'| + |W| - |\mathcal{S}(\mathcal{K}')|}{|W|} \right\rceil & \text{if } W, \mathcal{K}' \text{ satisfy conditions of Lemma 6.3 for } \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$$

$$\texttt{LB}^w_{\mathcal{I}}(W, \mathcal{K}') := \begin{cases} \left\lceil \frac{|\mathcal{K}'|}{|W|} \right\rceil & \text{if } W, \mathcal{K}' \text{ satisfy conditions of Lemma 6.3 for } \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$$

We will drop the subscript $\mathcal{I}$ when the instance is clear from context. Note that $\texttt{LB}^w$ is a weaker bound, in that it does not include any arcs due to connectivity within W. Since it holds $|W| - |\mathcal{S}(\mathcal{K}')| \geq 0$ for all inputs W and $\mathcal{K}'$, we have the following corollary.

Corollary 6.4. *Suppose $\mathcal{I} = (D, \mathcal{K})$ is a tree instance. For all $W \subseteq N$ and $\mathcal{K}' \subseteq \mathcal{K}$,*

$$\Delta^* \geq \mathtt{LB}^w(W, \mathcal{K}').$$

In the single-source setting, we will prove that for each instance $\mathcal{I}$, Δ^* is equal to $\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \mathtt{LB}(W, \mathcal{K}')$. (With a single source, D necessarily forms an out-tree with root s.) A natural next step is to ask if the lower bound construction is also exact for multi-source out-tree instances. However, this is not the case, since there are instances of `min-degree-SPP` on out-trees where $\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \mathtt{LB}(W, \mathcal{K}') = \Delta^* - 1$. Consider the out-tree instance on the right, where there are two sources s_1 and s_2. The remaining nodes are sinks labelled with each of the corresponding indices of sources that are matched to it. That is, a node v labelled with $\{1, 2\}$ indicates that there are commodities (s_1, v) and (s_2, v). In this instance, $\Delta^* = 3$, whereas $\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \mathtt{LB}(W, \mathcal{K}') = 2$.

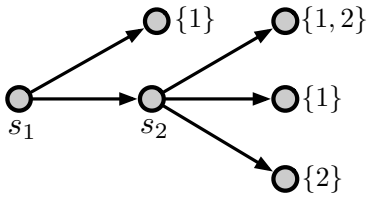


Figure 6.4: Out-tree instance where $\mathtt{LB}^w(W, \mathcal{K}') = \Delta^* - 1$.

While the witness set construction is not exact, it remains strong for out-tree instances. We prove in Section 6.4 that for any (multi-source) out-tree instance, even the weaker bound has a gap of at most one:

$$\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \mathtt{LB}^w(W, \mathcal{K}') \geq \Delta^* - 1.$$

We establish this result by presenting a polynomial-time algorithm for out-trees with multiple sources that returns a feasible solution with max out-degree at most one more than optimal. The single-source and multi-source algorithms are purely combinatorial and the analysis relies on bounds via witness sets.

6.3 Simple algorithm for single-source setting

In this section we present a fast combinatorial algorithm for the single-source setting. We first show that this setting reduces to the problem of min-degree arborescence in a directed acyclic graph, which can be solved by bipartite b-matching in $O(n^{2/3} m \log n)$ time [22] or a local search algorithm in $O(nm \log n)$ time [66]. We then present, in detail, the simple and fast combinatorial algorithm for single-source tree instances of `min-degree-SPP` that performs in linear time ignoring log factors. Moreover, we show that for each single-source instance $\mathcal{I}$, $\Delta^* = \max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \mathtt{LB}(W, \mathcal{K}')$. While our algorithm has similarities to the one

presented in [66], we can exploit the structure of transitive closure to reduce the number of arc swaps. We show that an efficient implementation of our approach runs in $O(n \log^2 n)$ time, a significant improvement over previous results.

Lemma 6.5. *Single-source tree instances of* `min-degree-SPP` *reduce to the problem of finding a min-degree arborescence in a directed acyclic graph.*

Proof. Let $\mathcal{I} = (D, \mathcal{K})$ be a tree instance of `min-degree-SPP` with a single source, s. Note that D is an out-tree rooted at s, and $\texttt{cl}(D)$ is a directed acyclic graph. We may assume that each leaf node in D is a sink for some commodity, as otherwise this node could be removed.

It suffices to prove that there is an optimal solution that contains an s, v-dipath for each node $v \in N$. Let H be an arbitrary optimal solution. Certainly if $v \in \mathcal{T}$, then an s, v-dipath is enforced by feasibility. So consider some node $v \notin \mathcal{T}$ and suppose there is no s, v-dipath in H. Then we may assume v has no out-arcs in H as otherwise they could be removed while maintaining feasibility.

Let T_v be the subtree in D rooted at v. Since each leaf is a sink for some commodity, there is some arc $e = uw$ in $\delta^+_H(N \setminus V(\mathcal{T}_v))$. We obtain an optimal solution with an s, v-dipath by removing arc uw and adding arcs uv and vw. Repeating this process gives an optimal solution that is an s-arborescence. □

In a directed acyclic graph, the min-degree arborescence problem can be cast as a b-matching problem in a bipartite graph. We create two copies of each node v in N, placing one copy, v^{out}, in a set A and the other copy, v^{in} in a set B. The arc set is formed by adding an arc $v^{out}w^{in}$ for each arc vw in $\texttt{cl}(D)$. Given a bound T on the maximum out-degree, we set the degree bound b_v to T for each node $v \in A$, and 1 for each node $v \in B$. Since D is acyclic, for a given value of T, the corresponding b-matching problem has a solution E of size $n - 1$ if and only if the corresponding edges in $\texttt{cl}(D)$ form an arborescence with max out-degree T. To obtain an optimal solution, we can run a black-box b-matching algorithm for $\log n$ many candidate values of T, since $\Delta^* \leq n - 1$. Since b-matching can be solved in $O(n^{2/3}m)$ time [22], this gives a solution in $O(n^{2/3}m \log n)$ time.

In the remainder of this section, we describe the simple combinatorial algorithm that solves single-source instances. First, we show that in the single-source setting, we can simplify the construction of witness sets. This set of lower bounds allows us to look at the base graph D rather than keeping track of the commodity set. Specifically, we show that we can

replace $|\mathcal{K}'|$ with $|\delta_D^+(W)|$ in Corollary 6.3 by recalling that all nodes v in D with $\delta_D^+(v) = \emptyset$ must be the sink of some commodity.

Corollary 6.6. *Suppose $\mathcal{I} = (D, \mathcal{K})$ is a tree instance and $|\mathcal{S}| = 1$. Let $W \subseteq N$ such that $s \in W$ and $D[W]$ is weakly-connected. Then*

$$\Delta^* \geq \left\lceil \frac{|\delta_D^+(W)| + |W| - 1}{|W|} \right\rceil.$$

Proof. It suffices to show that for any such node set W, there is a set of commodities $\mathcal{K}'$ with cardinality $|\delta_D^+(W)|$ such that for each $k \in \mathcal{K}'$, $t_k \notin W$, and for all distinct $k, j \in \mathcal{K}'$, $\delta_{P_k}^+(W) \cap \delta_{P_j}^+(W) = \emptyset$.

Consider an arc $e \in \delta_D^+(W)$. Since each node v with $\deg_D^+(v) = 0$ is a sink for some commodity $k \in \mathcal{K}$, there must be a commodity k such that $e \in P_k$, and more specifically, $e = \delta_{P_k}^+(W)$. For each arc $e \in \delta_D^+(W)$, let k_e denote an arbitrary commodity such that $e = \delta_{P_{k_e}}^+(W)$, and let $\mathcal{K}' = \bigcup_{e \in \delta_D^+(W)} \{k_e\}$. Observe that for each pair of edges e and f in $\delta_D^+(W)$, $\delta_{P_{k_e}}^+(W) \cap \delta_{P_{k_f}}^+(W) = \emptyset$. Thus, $\mathcal{K}'$ satisfies the desired conditions. □

Let $\mathcal{I} = (D, \mathcal{K})$ be an instance of `min-degree-SPP`. Recall that for each $W \subseteq N$ and $\mathcal{K}' \subseteq \mathcal{K}$,

$$\texttt{LB}_{\mathcal{I}}(W, \mathcal{K}') := \begin{cases} \left\lceil \frac{|\mathcal{K}'| + |W| - 1}{|W|} \right\rceil & \text{if } W, \mathcal{K}' \text{ satisfy the conditions of Lemma 6.3 for } \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$$

We define the following function.

$$\texttt{LB}_{\mathcal{I}}(W) := \begin{cases} \left\lceil \frac{|\delta_D^+(W)| + |W| - 1}{|W|} \right\rceil & \text{if } W \text{ satisfies the conditions of Corollary 6.6 for } \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$$

Again, when the instance $\mathcal{I}$ is clear from context, we drop the subscript. Observe that the two families of lower bounds have equal strength since for any choice of $W \subseteq N$, the subset $\mathcal{K}'$ that maximizes the value of $\texttt{LB}(W, \mathcal{K}')$ has size $|\delta^+(W)|$. That is,

$$\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \texttt{LB}(W, \mathcal{K}') = \max_{W \subseteq N} \texttt{LB}(W).$$

We will prove that for single-source instances, Δ^* is equal to $\max_{W \subseteq N} \texttt{LB}(W)$ (Proposition 6.7). Moreover, we will prove that $\operatorname{argmax}_{W \subseteq N} \texttt{LB}(W)$ can be found in polynomial time as a byproduct of a exact polytime algorithm for `min-degree-SPP` in the single-source setting.

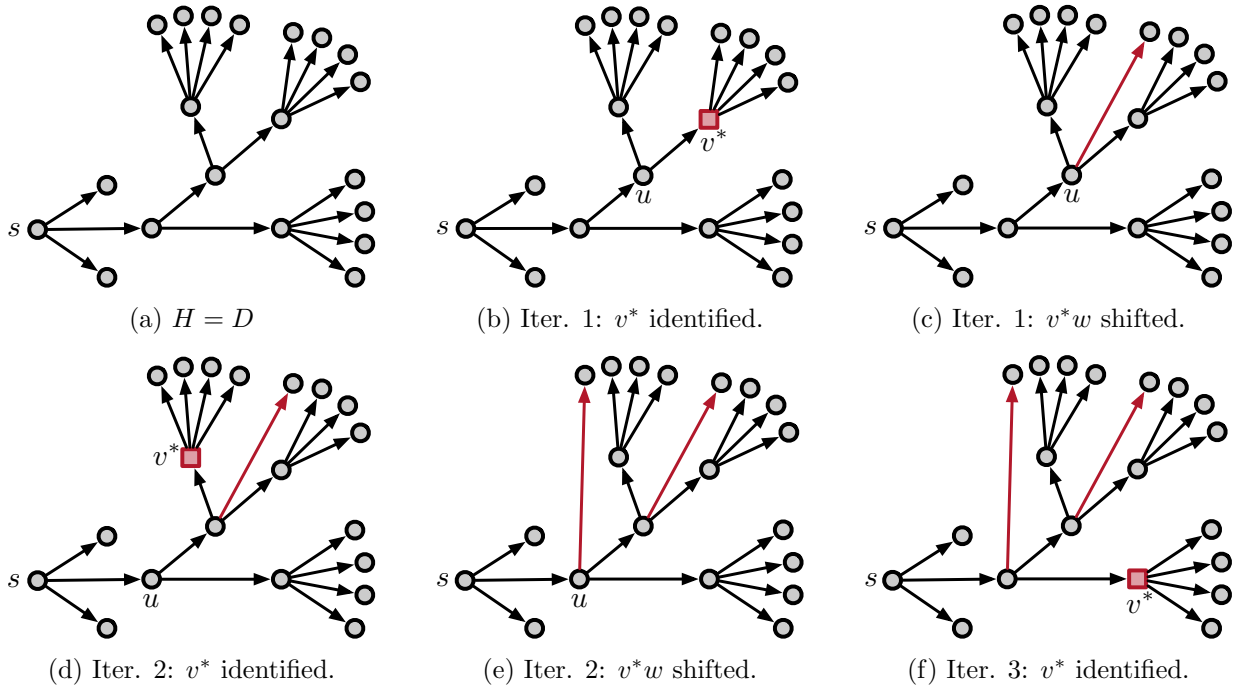
6.3.1 Single-source algorithm

In the single-source setting, D is a directed out-tree rooted at the source s. The algorithm can be described as follows. We begin with the feasible subgraph H_0 set to D. Let H_i denote the feasible subgraph obtained in iteration i. In iteration i, we identify a node in H_{i-1} with the highest out-degree, denoted v^*. We then attempt to shift an arc v^*w in H_{i-1} to instead depart a node along the s, v^*-dipath in D with out-degree at most $\Delta^+(H_{i-1}) - 2$ in H_{i-1}. If such a node exists, we let u denote the nearest such node to v^* in D, and define H_i as the subgraph obtained from H_{i-1} by replacing arc v^*w with arc uw. If no such node exists, the algorithm terminates with $H = H_{i-1}$. This procedure is restated in Algorithm 14, where P_{uv} is used to denote the unique dipath in D from u to v. In the following algorithm, we define a *topological ordering*, $\preceq$, as an ordering of the nodes so that for any arc $uv \in A$, $u \prec v$.

Algorithm 14: Local search algorithm for the single-source setting.

```
Assign a topological ordering ⪯ to the nodes
H_0 ← D
i = 1
while True do
    Let v* ∈ argmax_{v∈N} deg^+_{H_{i-1}}(v)
    Let v*w ∈ δ^+_{H_{i-1}}(v*)
    R ← {u ∈ V(P_{sv*}) : deg^+_{H_{i-1}}(u) ≤ deg^+_{H_{i-1}}(v*) − 2}
    if R ≠ ∅ then
        Let u ∈ R such that y ⪯ u for all y ∈ R
        H_i ← H_{i-1} \ {v*w} ∪ {uw}
    else
        return H = H_{i-1}
    i = i + 1
```

Figure 6.5 demonstrates the steps of the algorithm when the base graph D is as provided in Figure 6.5a, and $\mathcal{T}$ is the set of leaves. The square node is the selected max out-degree node in each iteration. Observe that in iteration 3, the set R is empty and so the algorithm terminates.



(a) $H = D$ (b) Iter. 1: v^* identified. (c) Iter. 1: v^*w shifted.

(d) Iter. 2: v^* identified. (e) Iter. 2: v^*w shifted. (f) Iter. 3: v^* identified.

Figure 6.5: Execution of local search algorithm for single-source setting.

The following observations are used in arguing for the efficiency and optimality of the local search algorithm.

(1) If a vertex u has $\deg^+_{H_i}(u) \geq \Delta^+(H_i) - 1$ in iteration i, then in each subsequent iteration $j \geq i$, $\deg^+_{H_j}(u) \geq \Delta^+(H_j) - 1$;

(2) Subgraph H_i is an s-arborescence in each iteration.

The first statement holds since in each iteration, the only node that decreases in out-degree is the identified max out-degree node. Roughly, (1) states that once a node has high out-degree relative to other nodes in some iteration, it may decrease in out-degree, but it remains a high out-degree node relative to the other nodes in each iteration. The second observation follows from the fact that D is directed tree, and in each iteration H remains a single connected component without increasing the number of arcs.

Proposition 6.7. *Algorithm 14 returns a feasible subgraph H with $\Delta^+(H) = \Delta^*$ given an instance with a single source in $O(n^2 \log n)$ time.*

Proof. First, we argue that the algorithm terminates in polynomial time.

In each iteration, the out-degree of some maximum out-degree node is reduced by 1. Given a maximum out-degree of Δ, there are at most $\lfloor \frac{n-1}{\Delta} \rfloor$ iterations until the maximum out-degree is reduced to $\Delta - 1$. Since in the first iteration $\Delta \leq n-1$, the total number of iterations is upper bounded by

$$\sum_{\Delta=1}^{n-1} \left\lfloor \frac{n-1}{\Delta} \right\rfloor \leq n \sum_{\Delta=1}^{n} \frac{1}{\Delta} = O(n \log n).$$

Since each iteration can be implemented in $O(n)$ time, the algorithm runs in $O(n^2 \log n)$ time.

It remains to argue that the subgraph H produced by the algorithm is both feasible and $\Delta^+(H) = \Delta^*$. The feasibility of H follows from the observation that in each iteration, for each node $v \in N$, the current subgraph contains an s, v-dipath.

We now prove that $\Delta^+(H) = \Delta^*$, by presenting a witness set W such that $\texttt{LB}(W) = \Delta^+(H)$. Let v^* be a max out-degree node in H. We iteratively construct W as follows. First, $W = V(P_{sv^*})$. Note that for all $v \in V(P_{sv^*})$, $\deg^+_H(v) \geq \Delta^+(H) - 1$ since the algorithm terminated. Then, while there is a node $u \notin W$ such that in some iteration an arc uw was exchanged for an arc vw where $v \in W$, we add $V(P_{su})$ to W. Note that if such an exchange of arcs occurred, u was a max out-degree node in some iteration i, and all nodes along the v, u-dipath, excluding v, had out-degree at least $\Delta^+(H_i) - 1$ in H_i. By observation $(P1)$, it follows that each such node has out-degree at least $\Delta^+(H) - 1$ in H. Thus, for each node $v \in W$, $\deg^+_H(v) \geq \Delta^+(H) - 1$. Figure 6.6 shows the node set W obtained for the instance solving in Figure 6.5.

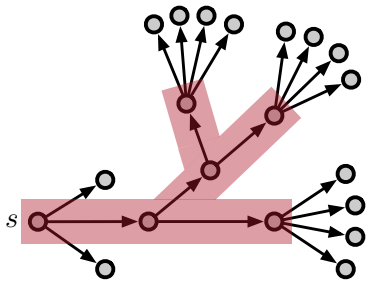


Figure 6.6: The set W certifies optimality of H from Figure 6.5f.

Observe that $D[W]$ is weakly-connected since W is formed by adding the node set of dipaths from s in D. As a result,

$$|\delta^+_H(W)| \geq (\Delta^+(H) - 1)|W| + 1 - (|W| - 1) = (\Delta^+(H) - 2)|W| + 2.$$

Furthermore, W contains s and $\delta_D^+(W) \neq \emptyset$, so W satisfies the conditions of Corollary 6.6. Thus, it suffices to show that $|\delta_D^+(W)| \geq |\delta_H^+(W)|$, since then

$$\texttt{LB}(W) = \left\lceil \frac{|\delta_D^+(W)| + |W| - 1}{|W|} \right\rceil \geq \left\lceil \frac{(\Delta^+(H) - 2)|W| + 2 + |W| - 1}{|W|} \right\rceil \geq \Delta^+(H)$$

Let $vw \in \delta_H^+(W)$. This arc corresponds to the unique arc entering w in D, i.e., uw (where possibly $u = v$). By the iterative construction of W, $u \in W$, and so $uw \in \delta_D^+(W)$. Furthermore, no two arcs in $\delta_H^+(W)$ correspond to the same arc in $\delta_D^+(W)$ since throughout the algorithm, the feasible subgraph remains an arborescence (and hence the total number of arcs is unchanged). If a pair of arcs in $\delta_H^+(W)$ were obtained via exchanges of arcs originating at the same arc uw in $\delta_D^+(W)$, then at some point in the algorithm the number of arcs would have decreased. Thus, $|\delta_D^+(W)| \geq |\delta_H^+(W)|$ as required. □

To obtain a more efficient algorithm for the single-source setting, we can reduce the number of operations by assuming we are given a target max out-degree, T. Consider a node v with *more* than T nodes in $N_D^+(v)$, and let $\alpha_v = |N_D^+(v)| - T$ denote the number of excess nodes. If there is a solution with $\Delta^+(H) \leq T$, at least α_v nodes in the set $N_D^+(v)$ must instead be reached by arcs departing a predecessor of v in D. We divide the set $N_D^+(v)$ into a set of *fixed nodes*, F_v, of cardinality $\min\{|N_D^+(v)|, T\}$, and the *remaining nodes* $R_v = N_D^+(v) \setminus F_v$. We then generate a solution H, if $T \geq \Delta^*$, so that H contains an arc vw for all $w \in F_v$, and an arc uw for all $w \in R_v$, for some predecessor u of v in D.

We store the set of descendant nodes of a node v that have not yet been allocated in a set D_v, which is built throughout the algorithm. In other words, D_v is the set of remaining nodes that v inherited from its descendants that have not yet been allocated to a node. Moving from the leaves towards the root, s, we shift any remaining nodes in R_v and D_v, to the set $D_{p(v)}$, where $p(v)$ denotes the parent of v in D. If we encounter a node with *fewer* than T nodes in F_v, we then take a maximum of $-\alpha_v$ nodes from the set D_v and allocate them to v. Note that $-\alpha_v = T - |N_D^+(v)| > 0$ and $R_v = \emptyset$ for such nodes. Since the arcs departing the max out-degree nodes in Algorithm 14 were chosen arbitrarily, the correctness of the algorithm implies that this procedure will produce a feasible solution H with $\Delta^+(H) \leq T$, so long as $T \geq \Delta^*$. For the root node in D, s, let $p(s) := s$.

Algorithm 15: Target algorithm for the single-source setting.

Input: A target $T \in \mathbb{Z}_{\geq 1}$, and a single-source instance of `min-degree-SPP`, $\mathcal{I} = (D, \mathcal{K})$, where $D = (N, A)$ with root s.

$V \leftarrow \{s = v_1, v_2, \ldots, v_n\}$, in topological order
$F_v \leftarrow \min\{|N_D^+(v)|, T\}$ nodes from $N_D^+(v)$ $\forall v \in N$
$R_v \leftarrow N_D^+(v) \setminus F_v$ $\forall v \in N$
$D_v \leftarrow \emptyset$ $\forall v \in N$
$\alpha_v = |N_D^+(v)| - T$ for all $v \in N$
for $i = n$ *to* 1 **do**
    $v \leftarrow v_i$
    if $\alpha_v < 0$ **then**
        Let U be a set of $\min\{|D_v|, -\alpha_v\}$ nodes from D_v
        $D_v \leftarrow D_v \setminus U$
        $F_v \leftarrow F_v \cup U$
    $D_{p(v)} \leftarrow D_{p(v)} \cup R_v \cup D_v$
if $D_s \neq \emptyset$ **then**
    return $\emptyset$
else
    return $H = \{vw : w \in F_v, v \in N\}$

Proposition 6.8. *Algorithm 15 runs in $O(n \log n)$ time, and returns a solution H with $\Delta^+(H) \leq T$ whenever $T \geq \Delta^*$.*

Proof. The correctness of the algorithm follows directly from that of Algorithm 14.

In the first step, the topological ordering can be computed in $O(n)$ time. In this process, for each node, we create an object that stores the parent node. The sets of nodes F_v, R_v, and D_v as well as the value of α_v are computed in $O(n)$ time. Producing H given the sets F_v also takes $O(n)$ time. It remains to argue that the for-loop can be executed in $O(n \log n)$ operations.

This can be argued via *union-find* type arguments (e.g., see Lecture 10 in [38]). We store the sets F_v, R_v, and D_v as linked lists, with each element storing both a pointer to the next element, as well as a pointer to the head of its list. The union of two lists of length L_A and L_B takes $O(\min(L_A, L_B))$ time, by splicing the shorter list into the longer list. Then, each element x has its head pointer updated at most $\log n$ times.

Our algorithm has the added complication that elements are *removed*. Since we remove an arbitrary set of nodes for a fixed size, $-\alpha$, this can be done in $O(-\alpha)$, since we can select the first $-\alpha$ nodes after the head of the list (or the entire list). Furthermore, each node is deleted at most once, and so the total operations for deletion is $O(n)$.

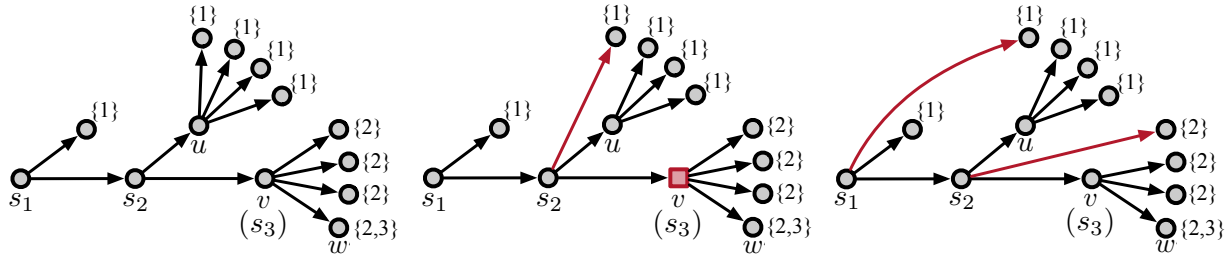
To maintain that element x has its head pointer updated at most $\log n$ times in the union operations, instead of splicing the shorter of the two sets into the larger one, we measure the lengths of the sets *as though no deletions have occurred*. Therefore the runtime of the algorithm is $O(n \log n)$ as claimed. □

Theorem 6.9. *There is an $O(n \log^2 n)$-time exact algorithm for tree instances of* `min-degree-SPP` *with a single source.*

Proof. By executing Algorithm 15 for $\log n$ many possible target values, we determine the smallest value of T for which Algorithm 15 gives a feasible subgraph H with $\Delta^+(H) \leq T$. Since Algorithm 15 did not return a feasible subgraph for the target $T - 1$, $\Delta^* \geq T$. Therefore, $\Delta^+(H) = \Delta^*$. □

6.4 Additive 1-approximation algorithm for out-trees

First, we describe how the multi-source setting differs from the single-source setting, making an extension of Algorithm 14 non-trivial. Consider the instance given in Figure 6.7a. The sinks are the set of leaves, and each sink t is labelled with the set of indices of sources for which there is a commodity with that source and sink t. Observe that we can no longer select departing arcs to shift arbitrarily: the arc vw cannot be shifted since there is a commodity with source v and sink w, while the other arcs departing v can be shifted. Furthermore, it is no longer the case that in order to decrease the degree of the highest node, we only need to shift a single arc. In Figure 6.7b we see that in the potential second iteration of Algorithm 14, no arc can be moved from v since s_2 has out-degree 3, and all arcs departing v serve commodities with source s_2.



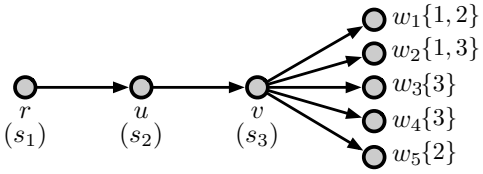
(a) Out-tree instance (b) Terminal stage for alg. (c) Solution ($\Delta^* = 3$).

Figure 6.7: Challenges in extending the single-source algorithm to multiple sources.

Definition 6.10 ($\mathcal{S}(a)$)**.** *Let $D = (N, A)$ be an out-tree, and let $a \in A$. The* set of sources that require a*, denoted $\mathcal{S}(a)$, is the set of sources s_k for which a is on the unique s_k, t_k-dipath in D. That is, $\mathcal{S}(a) := \{s_k : a \in A(P_k), k \in \mathcal{K}\}$.*

Definition 6.11 (Blocking source)**.** *Let $D = (N, A)$ be an out-tree and let $a = vw$ be an arc in A. The* blocking source *of a, denoted $b(a)$, is the unique source s in $S(a)$ such that the s, v-dipath in D has the fewest arcs.*

Consider the instance in Figure 6.8, where nodes w_i for $i \in [5]$ are sinks, and each is labelled with the corresponding set of indices of sources. For each arc a in D, the chart on the right indicates the set $\mathcal{S}(a)$ and value of $b(a)$.



a	$\mathcal{S}(a)$	$b(a)$
ru	$\{s_1\}$	s_1
uv	$\{s_1, s_2\}$	s_2
vw_1	$\{s_1, s_2\}$	s_2
vw_2	$\{s_1, s_3\}$	s_3
vw_3	$\{s_3\}$	s_3
vw_4	$\{s_3\}$	s_3
vw_5	$\{s_2\}$	s_2

Figure 6.8: Example of blocking sources

We will present an algorithm that takes as input a target, T, and returns a feasible solution H with $\Delta^+(H) \leq T$ whenever $T \geq \Delta^* + 1$. We now define the contraction subroutine used to generate sub-instances of `min-degree-SPP`.

Contraction of an instance for target T

Let $\mathcal{I} = (D, \mathcal{K})$ be a feasible out-tree instance of `min-degree-SPP` with root r, and let $T \in \mathbb{Z}_{>0}$. For any node $v \in N$, let $N_D^+(v)$ be the set of nodes reached by arcs departing v in D.

Suppose there is more than one non-leaf node. Let v be a non-leaf node where all nodes in $N_D^+(v)$ are leaves. That is, the subgraph of D rooted at v is a claw, denoted C_v. Such a node can be found efficiently by starting at r and moving to a descendant that is not a leaf. For any pair of nodes $v, w \in N$, let $d(v, w)$ denote the number of edges in the unique path between v and w in D. Breaking ties arbitrarily, let A_v^T denote the $\min\{T, |\delta_D^+(v)|\}$ arcs $a \in \delta_D^+(v)$ with the smallest values of $d(b(a), v)$. Let $B_v^T = \delta_D^+(v) \setminus A_v^T$. We write A_v and B_v when T is clear from context. In the example in Figure 6.8, we see that C_v is a claw, and when T is 3, $A_v = \{vw_2, vw_3, vw_4\}$ and $B_v = \{vw_1, vw_5\}$ since $b(vw_2), b(vw_3), b(vw_4) = s_3 = v$ and $b(vw_1), b(vw_5) = s_2$ which is further from v.

By definition, for any pair of arcs $a \in A_v$ and $a' \in B_v$, $b(a)$ is on the unique path in D between $b(a')$ and v. Let $V(A_v)$ denote the heads of the arcs in A_v and let $V(B_v)$ denote the heads of the arcs in B_v. For the same example and target, $V(A_v) = \{w_2, w_3, w_4\}$ and $V(B_v) = \{w_1, w_5\}$.

We define the instance obtained from $\mathcal{I}$ by *contracting v for target T*, denoted $\mathcal{I}_v^T = (D_v^T, \mathcal{K}_v^T)$ as follows. For all $v \neq r$, let $p(v)$ denote the parent node of v in D.

Definition 6.12. *The instance obtained from $\mathcal{I}$ by contracting v for target T is $\mathcal{I}_v^T = (D_v^T, \mathcal{K}_v^T)$, where*

$$E := \{p(v)w : vw \in B_v\},$$
$$D_v^T := \big(D \setminus \delta_D^+(v)\big) \cup E,$$

and the commodity set is $\mathcal{K}_v^T = \{(s_k, t'_k) : k \in \mathcal{K}\}$ where

$$t'_k = \begin{cases} v, & \text{if } t_k \in V(A_v) \\ t_k, & \text{otherwise.} \end{cases}$$

An example of this contraction process is given in Figure 6.9. The instance is given on the left, where the sinks are only labelled for the nodes in the claw C_v, since this is the only set of commodities impacted by the contraction of v. $\mathcal{I}_v^3$ is given on the right.

Recall that for a node $v \in N$, $\mathcal{T}(v)$ denotes the set of sinks among commodities with source v. That is, $\mathcal{T}(v) = \{t_k : (v, t_k) \in \mathcal{K}\}$.

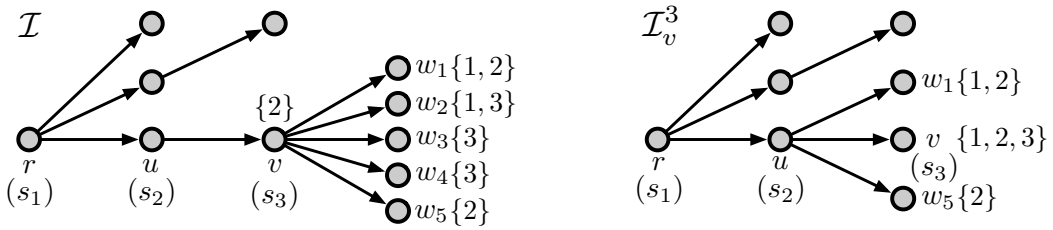


Figure 6.9: Example for contraction process.

Lemma 6.13. *If $T \geq |\mathcal{T}(v)|$, then $\mathcal{I}_v^T$ is a feasible instance of* `min-degree-SPP`.

Proof. We need to show that for each commodity $k \in \mathcal{K}_v^T$, there is an s_k, t'_k-dipath in D_v^T. If $t'_k \notin V(C_v)$, then $t'_k = t_k$, and (s_k, t_k) was a commodity in $\mathcal{I}$. The s_k, t_k-dipath in D did not use any nodes in C_v and so it remains in D_v^T.

If $t'_k = v$, then there is some commodity (s_k, t_k) in $\mathcal{I}$ where $t_k \in V(C_v)$. Since $s_k \neq t_k$, the s_k, t_k-dipath in D consists of an s, v-dipath as a subpath. This dipath is unchanged in D_v^T.

Finally, suppose $t'_k \in V(C_v) \setminus v$. By construction, $t_k \in V(B_v)$. Then $t'_k = t_k$, and (s_k, t_k) is a commodity in $\mathcal{I}$, and D contains an s_k, t_k-dipath consisting of an s_k, v-dipath along with the arc vt_k. However, when forming D_v^T we remove the arc vt_k and replace it with the arc ut_k, and so the same dipath will not suffice. However, since $T \geq |\mathcal{T}(v)|$, it follows that $s_k \neq v$. Thus, the s_k, v-dipath is non-trivial and consists of an s_k, u-dipath along with the arc uv. Then, the s_k, u-dipath along with the arc ut_k forms an s_k, t_k-dipath in D_v^T as required. □

Algorithm

For a given target T, our algorithm returns a feasible solution H with $\Delta^+(H) \leq T$ whenever $T \geq \Delta^* + 1$. To guarantee the performance of the algorithm if no such solution is produced (the algorithm outputs the empty set), we show that in this case there is a witness set $W, \mathcal{K}'$ such that $\mathrm{LB}^w(W, \mathcal{K}') \geq T$.

When there is a single non-leaf node, r, either the graph itself is the desired solution with max out-degree at most T, or the set $W = \{r\}, \mathcal{K}' = \mathcal{K}$ is a witness set certifying that $\Delta^* \geq T$. Otherwise, we find a node, v, with only leaves as descendants. Then either the subtree rooted at v already provides a witness set, or we apply the target algorithm to the instance $\mathcal{I}_v^T$. If the algorithm produces a feasible solution H_v to $\mathcal{I}_v^T$ with $\Delta^+(H_v) \leq T$, we then extend this subgraph to a feasible solution H for $\mathcal{I}$ with $\Delta^+(H) \leq T$ by simply adding back the arcs in the set A_v. The pseudocode is presented in Algorithm 16.

Algorithm 16: `out-tree`$(\mathcal{I}, T)$

Input: A target $T \in \mathbb{Z}_{\geq 1}$, and an out-tree instance of `min-degree-SPP`, $\mathcal{I} = (D, \mathcal{K})$

$L \leftarrow \{v \in N : \delta_D^+(v) = \emptyset\}$
$I \leftarrow N \setminus L$
if $|I| = 1$ **then**
  if $|L| > T$ **then**
    return $\emptyset$
  return D
Let v be a non-leaf where $N_D^+(v) \subseteq L$
if $|\mathcal{T}(v)| > T$ **then**
  return $\emptyset$
$H_v \leftarrow$ `out-tree`$(\mathcal{I}_v^T, T)$
if $H_v = \emptyset$ **then**
  return $\emptyset$
else
  return $H_v \cup A_v$

If Algorithm 16 returns the empty set, we argue by induction that there is a witness set $W_v, \mathcal{K}'_v$ for the instance $\mathcal{I}_v^T$ such that $\mathrm{LB}^w_{\mathcal{I}_v^T}(W_v, \mathcal{K}'_v) \geq T$. We then extend this pair to a witness set $W, \mathcal{K}'$ for $\mathcal{I}$ such that $\mathrm{LB}^w_{\mathcal{I}}(W, \mathcal{K}') \geq T$. Note that we cannot necessarily set $W = W_v$ and $\mathcal{K}' = \mathcal{K}'_v$, since the commodity paths may differ in $\mathcal{I}$ and $\mathcal{I}_v^T$. For each commodity k, the corresponding source-sink pair is (s_k, t_k) in $\mathcal{I}$ and (s_k, t'_k) in $\mathcal{I}_v^T$. Let Q_k denote the unique s_k, t'_k-dipath in D_v^T, and recall that P_k denotes the unique s_k, t_k-dipath in D. The following lemmas relate the cut sets in $\mathcal{I}$ and $\mathcal{I}_v^T$ for a fixed commodity and node set.

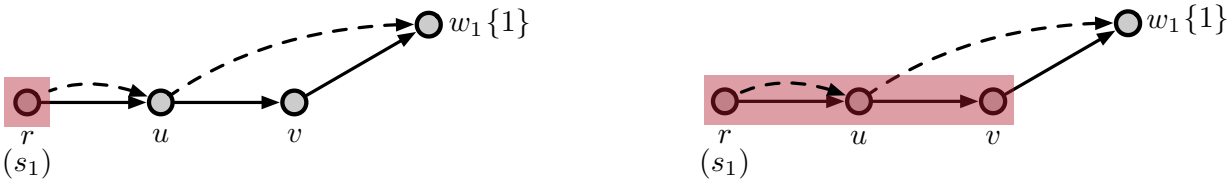
Lemma 6.14. *Let $X \subseteq N \setminus \{v\}$ such that $D[X]$ is weakly-connected, and let $k \in \mathcal{K}$. If $s_k \in X$ and $t'_k \notin V(B_v)$, then $\delta^+_{P_k}(X) = \delta^+_{Q_k}(X)$.*

Proof. If $t_k = v$, then $t'_k = t_k$ and P_k did not contain any arc from A_v or B_v. Therefore, P_k and Q_k are the same, and so $\delta^+_{P_k}(X) = \delta^+_{Q_k}(X)$. The same argument holds when $t'_k \neq v$.

Otherwise, $t'_k = v$ and $t_k \neq v$. It follows that $t_k \in V(A_v)$ and so Q_k is a subpath of P_k, giving $\delta^+_{Q_k}(X) \subseteq \delta^+_{P_k}(X)$. Since $v \notin X$ and $s_k \in X$, $\delta^+_{Q_k}(X) \neq \emptyset$. Moreover, $|\delta^+_{P_k}(X)| \leq 1$ in general, so $\delta^+_{P_k}(X) = \delta^+_{Q_k}(X)$. □

Lemma 6.15. *Suppose $|\mathcal{T}(v)| \leq T$. Let $X \subseteq N$ such that $D[X]$ is weakly-connected, and let $k \in \mathcal{K}$. If $s_k \in X$, $t'_k \in V(B_v)$, and $t'_k \notin X$, then either $\delta^+_{P_k}(X) = \delta^+_{Q_k}(X)$ or $\delta^+_{Q_k}(X) = p(v)t'_k$.*

Proof. Since $|\mathcal{T}(v)| \leq T$ and $t'_k \in V(B_v)$, s_k cannot be equal to v. Therefore, each of P_k and Q_k contain the same $s_k, p(v)$-dipath in D, denoted R, as a subpath. An example is shown in Figure 6.10, for the commodity with source r and sink w_1. The solid edges form P_k, and the dashed lines form Q_k. The subpath R is the dipath that is shared by P_k and Q_k (the single arc ru in this case).



(a) $\delta^+_R(X) \neq \emptyset$ (b) $\delta^+_R(X) = \emptyset$

Figure 6.10: Comparison of $\delta^+_{Q_k}(X)$ and $\delta^+_{P_k}(X)$.

If $\delta^+_R(X) \neq \emptyset$, then $\delta^+_{Q_k}(X) = \delta^+_R(X) = \delta^+_{P_k}(X)$. So suppose $\delta^+_R(X) = \emptyset$. Since $t'_k \notin X$, it follows that $\delta^+_{Q_k}(X) \neq \emptyset$. The only arc in $Q_k \setminus R$ is the arc $p(v)t'_k$. □

Proposition 6.16. *For each $T \in \mathbb{Z}_{>0}$, Algorithm 16 either returns a solution, H, to instance $\mathcal{I}$ with $\Delta^+(H) \leq T$, or there is a witness set $W, \mathcal{K}'$ such that $LB^w(W, \mathcal{K}') \geq T$, certifying that $\Delta^* \geq T$.*

Proof. Let $\mathcal{I} = (D, \mathcal{K})$ denote an instance of `min-degree-SPP` where D is an out-tree with root r. Let T be a positive integer. We prove the result by induction on the number of non-leaf nodes in D. Recall that we may assume that we are working with minimal, non-trivial instances, in the sense that all $k \in \mathcal{K}$ have $s_k \neq t_k$, $\mathcal{K} \neq \emptyset$, and any leaf node l must be a sink for some commodity.

Suppose there is a single non-leaf node, r, as in Figure 6.11. Then each node in $\delta^+_D(r)$ is a leaf. If $|\delta^+_D(r)| \leq T$, then the graph D is a solution with $\Delta^+(D) \leq T$. Otherwise, $|\mathcal{T}(r)| = |N^+_D(r)| > T$, and the witness set $W = \{r\}$, $\mathcal{K}' = \mathcal{K}$ gives the lower bound $\mathrm{LB}^w(W, \mathcal{K}') = \lceil |\mathcal{T}(r)|/1 \rceil \geq T$.

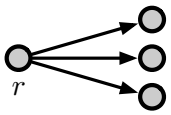


Figure 6.11: Single non-leaf node.

Suppose the result holds for all out-tree instances with at most d non-leaf nodes, and consider an instance with $d+1$ non-leaf nodes. Let v be a non-leaf node where all nodes in $N_D^+(v)$ are leaves.

If $|\mathcal{T}(v)| > T$, then setting $W = \{v\}$ and $\mathcal{K}' = \{k \in \mathcal{K} : s_k = v\}$ gives $\mathrm{LB}^w(W, \mathcal{K}') > T$, so suppose $|\mathcal{T}(v)| \leq T$. Let A_v, B_v, E, and $\mathcal{I}_v^T = (D_v^T, \mathcal{K}_v^T)$ be defined as above, where $A_v \cup B_v = \delta_D^+(v)$ and for any pair $a \in A_v$ and $a' \in B_v$, $b(a)$ is on the unique path in D between $b(a')$ and v. By Lemma 6.13, it follows that $\mathcal{I}_v^T$ is a feasible out-tree instance with at most d non-leaf nodes. Let Δ_v^* denote the min-max degree of a feasible subgraph of $\mathtt{cl}(D_v)$ for instance $\mathcal{I}_v^T$. By induction, the algorithm either returns a feasible subgraph $H_v \subseteq \mathtt{cl}(D_v)$ for $\mathcal{I}_v^T$ with $\Delta^+(H_v) \leq T$, or there is a witness set $W_v, \mathcal{K}_v'$ certifying that $\Delta_v^* \geq T$.

Suppose the algorithm returns a feasible subgraph $H_v \subseteq \mathtt{cl}(D_v)$ for $\mathcal{I}_v^T$ with $\Delta^+(H_v) \leq T$. Let $H = H_v \cup A_v$, as in the example in Figure 6.12. We claim that H is a feasible solution for $\mathcal{I}$ with $\Delta^+(H) \leq T$. Observe that $\deg_{H_v}^+(v) = 0$ and $|A_v| \leq T$, so $\Delta^+(H) \leq T$. It remains to argue that H is feasible. Let $k \in \mathcal{K}$. If $t_k \notin V(A_v)$, then (s_k, t_k) is a commodity in $\mathcal{I}_v^T$, and so an s_k, t_k-dipath is present in $H_v \subseteq H$. Otherwise $t_k \in V(A_v)$, and so there is a commodity (s_k, v) in $\mathcal{K}_v^T$. By feasibility of H_v, H contains an s_k, v-dipath. With the addition of arc vt_k in A_v we obtain an s_k, t_k-dipath in H. Therefore, H is feasible and $\Delta^+(H) \leq T$.

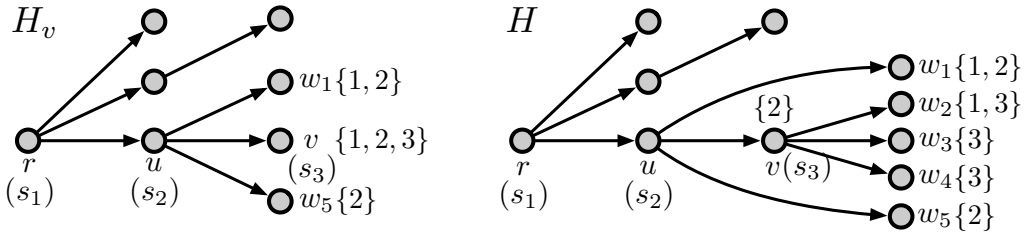


Figure 6.12: Extending feasible H_v for $\mathcal{I}_v^3$ to a feasible solution H for $\mathcal{I}$.

Suppose instead, there is a witness set $W_v, \mathcal{K}_v'$ with $\mathrm{LB}_{\mathcal{I}_v^T}^w(W_v, \mathcal{K}_v') \geq T$. That is, $D_v^T[W_v]$ is weakly-connected, $\emptyset \neq \mathcal{K}_v' \subseteq \mathcal{K}_v^T$ such that $s_k \in W_v$ and $t_k' \notin W_v$ for all $k \in \mathcal{K}_v'$. Moreover, for all distinct $k, j \in \mathcal{K}_v'$, $\delta_{Q_k}^+(W) \cap \delta_{Q_j}^+(W) = \emptyset$. Note that we may assume W_v does not

contain any leaves in D_v^T, since removing a leaf from W_v can only increase the lower bound. So, $W_v \subseteq N \setminus \{v\}$.

We now construct a witness set $W, \mathcal{K}'$ for $\mathcal{I}$ such that $\Delta^* \geq \mathrm{LB}^w_{\mathcal{I}}(W, \mathcal{K}') \geq T$. Consider the same set of nodes W_v and commodities $\mathcal{K}'_v$ in D instead of D_v^T. Since no leaf node in D_v^T is in W_v, $D[W_v]$ is weakly-connected. Moreover, for each $k \in \mathcal{K}'_v$, it holds $s_k \in W_v$ and $t_k \notin W_v$.

If $\delta^+_{P_k}(W) = \delta^+_{Q_k}(W)$ for all $k \in \mathcal{K}'_v$, then it follows that $W = W_v$ and $\mathcal{K}' = \mathcal{K}'_v$ gives $\mathrm{LB}^w_{\mathcal{I}}(W, \mathcal{K}') = \mathrm{LB}^w_{\mathcal{I}^T_v}(W, \mathcal{K}'_v) \geq T$ as required. So, suppose there is some commodity $k \in \mathcal{K}'_v$ such that $\delta^+_{P_k}(W) \neq \delta^+_{Q_k}(W)$. By Lemmas 6.14 and 6.15, it follows that there is a commodity $\ell \in \mathcal{K}'_v$ with $t'_\ell \in V(B_v)$ and $p(v) \in W_v$.

Let $W = W_v \cup \{v\}$. Since $p(v) \in W_v$ and $D[W_v]$ is weakly-connected, it follows that $D[W]$ is weakly-connected. Let $\bar{\mathcal{K}} = \{(b(vt_k), t_k) : t_k \in V(A_v)\}$. By definition of $\bar{\mathcal{K}}$ and B_v, it follows that s_k is on the unique dipath between s_ℓ and v for all $k \in \bar{\mathcal{K}}$. Since $D[W]$ is weakly-connected and contains both s_ℓ and v, it follows that $s_k \in W$ for all $k \in \bar{K}$.

If there is a commodity $k \in \mathcal{K}'_v$ with $t_k = v$, let k_v denote this commodity, and otherwise let k_v denote the empty set. Let $\mathcal{K}' = (\mathcal{K}'_v \cup \bar{\mathcal{K}}) \setminus k_v$. Then $s_k \in W$ for all $k \in \mathcal{K}'$, and $t_k \notin W$ for each $k \in \mathcal{K}'$ since k_v was removed. We also claim that $\delta^+_{P_k}(W) \cap \delta^+_{P_j}(W) = \emptyset$ for any $k, j \in \mathcal{K}'$. If $k \in \mathcal{K}'$ with $t_k \in N^+_D(v)$, then $\delta^+_{P_k}(W) = vt_k$. Otherwise, $t_k \notin V(C_v)$ and $\delta^+_{P_k}(W) = \delta^+_{Q_k}(W) \notin \delta^+_D(v)$. Therefore, the disjointness of the cuts follows.

In order to prove that $\mathrm{LB}^w_{\mathcal{I}}(W, \mathcal{K}') \geq T$, it remains to argue that the cardinality of $|\mathcal{K}'|$ is at least $(T-1)|W|+1$. Observe that by definition, if $B_v \neq \emptyset$, then $|A_v| = T$. By induction, we have $|\mathcal{K}'_v| \geq (T-1)|W_v|+1$. Therefore, $|\mathcal{K}'| \geq (T-1)|W_v|+T-1+1 = (T-1)|W|+1$, and so $\mathrm{LB}^w_{\mathcal{I}}(W, \mathcal{K}') \geq T$ as required. □

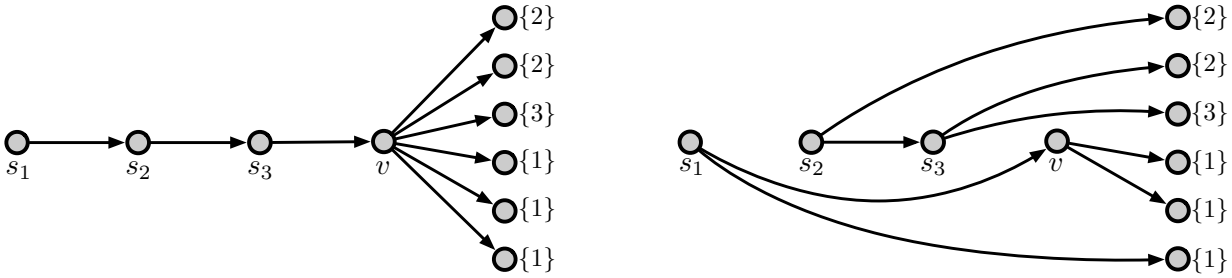
Theorem 6.17. *There is a polynomial-time additive* 1*-approximation algorithm for out-tree instances of* `min-degree-SPP`.

Proof. By executing Algorithm 16 for all possible target values, we determine the smallest value of T for which Algorithm 16 gives a feasible subgraph H with $\Delta^+(H) \leq T$. Since Algorithm 16 did not return a feasible subgraph for the target $T-1$, there is a witness set $W, \mathcal{K}'$ with $\mathrm{LB}^w(W, \mathcal{K}') \geq T-1$ (constructible in polytime), certifying that $\Delta^* \geq T-1$. Hence, $\Delta^+(H) \leq \Delta^* + 1$. □

As previously mentioned in the discussion of lower bounds, there are out-tree instances of `min-degree-SPP` where $\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \mathtt{LB}(W, \mathcal{K}') = \Delta^* - 1$. Furthermore, the analysis of

Algorithm 16 is tight since there are out-tree instances for which the algorithm does not return a solution with out-degree at most T, even if $\Delta^* = T$.

Consider the following instance in Figure 6.13a. A solution H with $\Delta^+(H) = 2$ is given in Figure 6.13b.



(a) Out-tree instance.

(b) Solution with $\Delta^+(H) = 2$.

Figure 6.13

However, when $T = 2$ in the target algorithm, in the first step we generate the subproblem given in Figure 6.14 by contracting node v for the target $T = 2$. The resulting problem does *not* have a feasible solution with max out-degree at most 2. Thus, the algorithm would not produce a feasible solution with max out-degree 2 for the original instance in Figure 6.13.

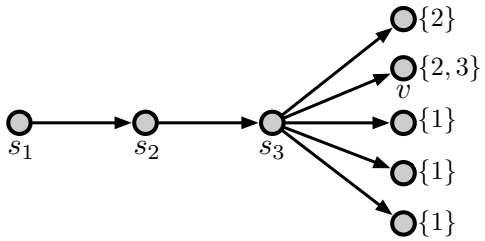


Figure 6.14

It remains open whether or not out-tree instances are NP-hard. In Section 6.5, we relate the hardness of these instances to that of a natural packing problem.

6.5 Towards out-tree hardness

It remains open whether or not out-tree instances of `min-degree-SPP` are NP-hard. Establishing hardness for this setting would demonstrate that our additive 1-approximation algorithm was the best possible in Section 6.4. Even in the seemingly simple case of *broom instances* – instances in which the underlying undirected graph of D is a broom – appears challenging. In this section we demonstrate that broom instances are at least as hard as a particular packing problem, so long as its inputs are polynomially bounded. We believe this packing problem is of independent interest, and establishing it is strongly NP-hard is one avenue to proving that `min-degree-SPP` is NP-hard on brooms, and out-trees more generally.

`SIGNED-VALUES-BIN-PACKING` (`SV-BP`): Given a set $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ of n integers, determine if there is a partition of $[n]$ into sets $N_1, N_2, \ldots, N_\ell$ for some $\ell \in [n]$ such that for each part $i \in [\ell]$, $\sum_{j \in N_i} w_j \leq 1$.

Note that we may assume that none of the input integers are 0 or 1, since $\mathcal{W}$ is a `YES`-instance if and only if $\mathcal{W}' = \{w_i : w_i \in \mathcal{W}, w_i \notin \{0, 1\}\}$ is a `YES`-instance. In the proof of Lemma 6.19, we establish that broom instances of `min-degree-SPP` are at least as hard as the problem of `SV-BP` when the integers are bounded by a polynomial in n. Thus, establishing that `SV-BP` is strongly NP-hard would prove that `min-degree-SPP` on brooms (and thus out-trees) is NP-hard.

Open problem 6.18. *Is* `SV-BP` *strongly NP-hard?*

Lemma 6.19. *Broom instances of* `min-degree-SPP` *are at least as hard as* `SV-BP` *with weights that are polynomial in* n*.*

Proof. We show that there is a reduction from `SV-BP` to broom instances of `min-degree-SPP`, when the integers are polynomial in n. Let $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ be the input integers of an instance of `SV-BP`, where the integers are polynomial in n, $w_1 \geq w_2 \geq \ldots, w_n$. We say that a partition of $\mathcal{W}$ is *valid* if each part sums to at most 1.

Let $T \geq 2$. We will construct a broom instance of `min-degree-SPP`, with sets of sources $S_1, S_2, \ldots, S_n$, so that there is solution with max out-degree T if and only if the weakly-connected sets of sources in an optimal solution corresponds to a valid partition of the index set $[n]$. We construct a corresponding instance of the broom problem with target T as follows.

Construction of broom instance, given $T \geq 2$: we create a set of sources, S_i, for each

index $i \in [n]$. In this set, we generate $\alpha_i + 1$ sources, where α_i is the smallest positive integer such that $w_i + \alpha_i(T-1) \geq 0$. Let $\beta_i := w_i + \alpha_i(T-1)$. We introduce $\beta_i + T - 1$ sinks, $T_i = \{t_i^1, t_i^2, \ldots, t_i^{\beta_i+T-1}\}$.

We construct a set $\mathcal{K}_i$ of commodities with sources in S_i and sinks in T_i as follows. The first source, s_i^1, is assigned β_i sinks. The final source, $s_i^{\alpha_i+1}$ is assigned the remaining $T-1$ sinks. Finally, we ensure that each source in S_i must have a path to a common sink node by defining a commodity with source s and sink $t_i^{\beta_i+1-1}$ for each source $s \in S_i$. Specifically, we have the following commodities.

- $\{(s_i^1, t_i^j) : j \in [\beta_i]\}$
- $\{(s_i^{\alpha_i+1}, t_i^j) : j \in [\beta_i + T - 1] \setminus [\beta_i]\}$
- $\{(s_i^j, t_i^{\beta_i+T-1}) : j \in [\alpha_i]\}$

Note that each node in $\{s_i^2, \ldots, s_i^{\alpha_i}\}$ is the source of only a single commodity.

The corresponding broom instance $\mathcal{I} = (D, \mathcal{K})$ has the digraph D as depicted in Figure 6.15, with a dipath on the source sets S_1 to S_n, and the sinks are the leaves. The commodity set is $\mathcal{K} = \cup_{i\in[n]}\mathcal{K}_i$.

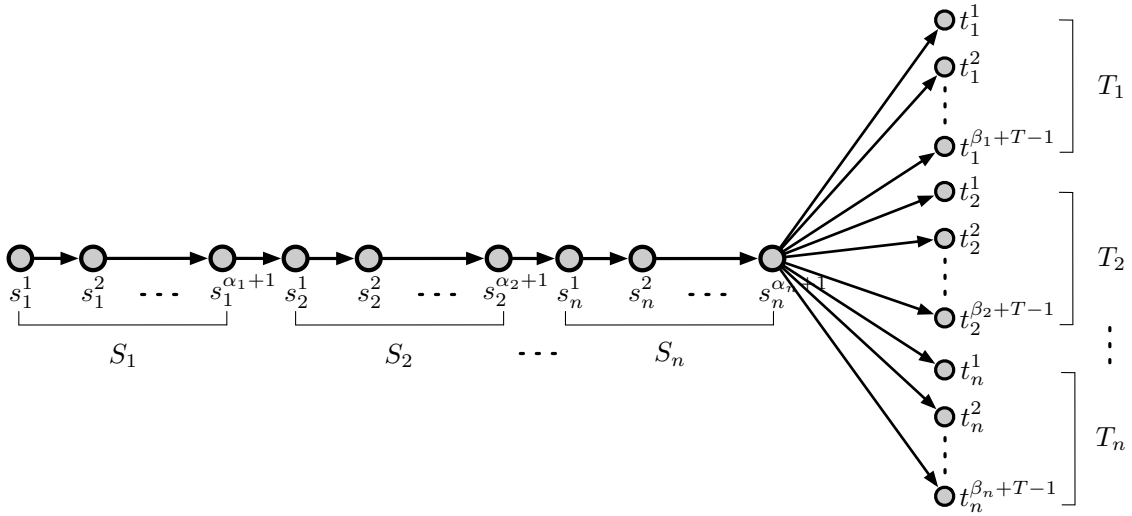


Figure 6.15: Digraph D for the broom instance.

We now prove that $\Delta^* \leq T$ if and only if there is a valid partition of $\mathcal{W}$.

($\Rightarrow$) Suppose there is a solution, H, to the instance $\mathcal{I} = (D, \mathcal{K})$ with $\Delta^+(H) \leq T$. The

solution splits into weakly-connected components, $H_1, H_2, \ldots, H_\ell$. Recall that each set of sources S_i will be in the same weakly-connected component due to the shared sink, $t_i^{\beta_i+T-1}$. For each component $j \in [\ell]$, let $N_j = \{i : S_i \in H_j\}$. We will prove that $N_1, N_2, \ldots, N_\ell$ is a valid partition of $\mathcal{W}$.

Let $\mathcal{S}_j$ and $\mathcal{T}_j$ denote the set of sources and sinks in component H_j respectively. By definition of N_j, it follows that $\mathcal{S}_j = \bigcup_{i \in N_j} S_i$ and $\mathcal{T}_j = \bigcup_{i \in N_j} T_i$. Furthermore, the node set of H_j is $\mathcal{S}_j \cup \mathcal{T}_j$. The fewest arcs possible for the subgraph H_j, since it is weakly-connected, is $|\mathcal{S}_j| + |\mathcal{T}_j| - 1$. Since every arc in H_i departs a node in $\mathcal{S}_j$, $\Delta^+(H_j) \leq T$ implies that $\frac{|\mathcal{S}_j|+|\mathcal{T}_j|-1}{|\mathcal{S}_j|} \leq T$. Therefore,

$$
\begin{aligned}
& |\mathcal{S}_j| + |\mathcal{T}_j| - 1 \leq T|\mathcal{S}_j| \\
\iff & \sum_{i \in N_j} (\alpha_i + 1) + \sum_{i \in N_j} (\beta_i + T - 1) - 1 \leq T \sum_{i \in N_j} (\alpha_i + 1) \\
\iff & \sum_{i \in N_j} \alpha_i + |N_j| + \sum_{i \in N_j} (w_i + \alpha_i(T-1)) + (T-1)|N_j| - 1 \leq T \sum_{i \in N_j} \alpha_i + T|N_j| \\
\iff & \sum_{i \in N_j} \alpha_i + \sum_{i \in N_j} (w_i + \alpha_i(T-1)) - 1 \leq T \sum_{i \in N_j} \alpha_i \\
\iff & \sum_{i \in N_j} \alpha_i + \sum_{i \in N_j} w_i + T \sum_{i \in N_j} \alpha_i - \sum_{i \in N_j} \alpha_i - 1 \leq T \sum_{i \in N_j} \alpha_i \\
\iff & \sum_{i \in N_j} w_i \leq 1.
\end{aligned}
$$

Since this holds for each $j \in [\ell]$, the partition $N_1, N_2, \ldots, N_\ell$ satisfies $\sum_{i \in N_j} w_i \leq 1$ for each $j \in [\ell]$.

($\Leftarrow$) Suppose there is a valid partition, $N_1, N_2, \ldots, N_\ell$, for $\mathcal{W}$. We will form a solution H to the broom instance $\mathcal{I} = (D, \mathcal{K})$, consisting of ℓ connected components, $H_1, H_2, \ldots, H_\ell$, where each component has maximum out-degree at most T.

Construction of H_j. Let $j \in [\ell]$ and let $\mathcal{S}_j = \bigcup_{i \in N_j} S_i$ and $\mathcal{T}_j = \bigcup_{i \in N_j} T_i$. First, H_j is formed by taking the unique dipath in $\texttt{cl}(D)$ on the nodes in $\mathcal{S}_j$. Let $\mathcal{P}_j$ denote this path. It remains to add arcs to each of the sinks in $\mathcal{T}_j$ so that there is an s_k, t_k-dipath for each commodity $k \in \mathcal{K}_i$, for each $i \in N_j$, while ensuring that $\Delta^+(H_j) \leq T$.

Let $\{s_1, s_2, \ldots, s_{|\mathcal{S}_j|}\}$ be the set of sources in $\mathcal{S}_j$, ordered so that there is an s_i, s_j-dipath in D whenever $i \leq j$. Let $\{t_1, t_2, \ldots, t_{|\mathcal{T}_j|}\}$ be the set of sinks in $\mathcal{T}_j$, ordered first by increasing subscript, and then increasing superscript (top to bottom from Figure 6.15).

We add arcs to H_j from $\mathcal{S}_j$ to $\mathcal{T}_j$ as follows. First we add an arc from $s_{|\mathcal{S}_j|}$ to each of the final T sinks in $\mathcal{T}_j$. In decreasing order of sources, we continue to add arcs from the current sink to the last $T-1$ sinks remaining in $\mathcal{T}_j$, until no sinks remain, or we run out of sources. Observe that the resulting graph, H_j, has $\Delta^+(H_j) \leq T$.

If $|\mathcal{T}_j| \leq (T-1)|\mathcal{S}_j| + 1$, then H_j will include all sinks in $\mathcal{T}_j$. This is indeed the case, since

$$|\mathcal{T}_j| \leq (T-1)|\mathcal{S}_j| + 1 \iff |\mathcal{T}_j| + |\mathcal{S}_j| - 1 \leq T|\mathcal{S}_j| \iff \sum_{i \in N_j} w_i \leq 1,$$

as proven in the previous direction.

Thus, it remains to show that for each commodity $k \in \mathcal{K}$ with $s_k \in \mathcal{S}_j$ and $t_k \in \mathcal{T}_j$, there is an s_k, t_k-dipath in H_j. We prove the following claims, and recall that we may assume all weights w_i are not equal to 0 or 1.

Claim 1: If $w_i \geq 2$, then $|T_i| \geq |S_i|T$.

> *Proof.* If $w_i \geq 2$, then $\alpha_i = 1$ and $|S_i| = 2$. Furthermore, it holds that $\beta_i = w_i + (T-1) \geq T+1$, and so $|T_i| = \beta_i + T - 1 \geq 2T$. Therefore, $|T_i| \geq |S_i|T$.

Claim 2: If $w_i \leq 0$, then $|T_i| \leq |S_i|(T-1)$.

> *Proof.* If $-(T-1) \leq w_i \leq 0$, then $\alpha_i = 1$ and $|S_i| = 2$. Furthermore, it must be that $0 \leq \beta_i \leq T-1$ and so $|T_i| = \beta_i + T - 1 \leq 2T - 2$. Thus, it is $|T_i| \leq |S_i|(T-1)$.
>
> If instead $w_i < -(T-1)$, then $\alpha_i \geq 2$, and $|S_i| \geq 3$. Furthermore, it holds $\beta_i \leq T$, and so $|T_i| \leq 2T - 1$. Note that for all $T \geq 0$, we have $2T - 1 \leq 3(T-1)$. Thus, $|T_i| \leq 3(T-1) \leq |S_i|(T-1)$.

For a contradiction, suppose there is some commodity with source s_p and sink t_q (indices from ordered nodes in $\mathcal{S}_j$ and $\mathcal{T}_j$), such that H_j contains no s_p, t_q-dipath. Moreover, among all commodities of this form, assume that p is the largest possible index. Since there is no s_p, t_q-dipath, the arc from $\mathcal{S}_j$ to t_q departs a node s_r with $r < p$. Similarly, since there is no s_p, t_q-dipath in H_j, it follows that

$$|\bigcup_{r \geq p} \mathcal{T}(s_r)| > (|\mathcal{S}_j| - p + 1)(T-1) + 1. \tag{6.1}$$

Since $|\mathcal{T}(s_i^r)| \leq T - 1$ whenever $r \geq 2$, and p is the largest index such that there is no s_p, t dipath for some $t \in \mathcal{T}(s_p)$, it follows that $s_p = s_i^1$ for some $i \in N_j$. Thus, an equivalent

statement to (6.1) is

$$\sum_{r \in N_j : r \geq i} |T_r| > 1 + \sum_{r \in N_j : r \geq i} |S_r|(T-1). \tag{6.2}$$

Due to the ordering of the sets S_1 to S_n in D and by Claim 2, inequality (6.2) cannot be satisfied if $w_i \leq 0$. Alternatively, $w_i \geq 2$. By Claim 1 and the ordering of the source sets,

$$\sum_{r \in N_j : r < i} |T_r| \geq \sum_{r \in N_j : r < i} |S_r|T. \tag{6.3}$$

But then by combining inequalities (6.2) and (6.3), we see that $|\mathcal{T}_j| > |\mathcal{S}_j|(T-1)+1$, a contradiction. Therefore, H_j is feasible as claimed, and $\Delta^* \leq T$. □

6.6 Arbitrary trees

In this section, we provide a framework for obtaining approximation results for arbitrary tree instances. As a first step, we give an efficient 2-approximation when D is a star. Our framework for obtaining an approximation algorithm for arbitrary tree instances builds on the approximability of junction tree instances, a generalization of stars. We also highlight the weakening strength of the witness set construction for more complex graphs.

6.6.1 Star instances

We have the following 2-approximate solution for star instances.

Proposition 6.20. *Given a star instance, $H = \{s_k t_k : k \in \mathcal{K}\}$ is a feasible solution with $\Delta^+(H) \leq 2\Delta^*$.*

Proof. Let $H = \{s_k t_k : k \in \mathcal{K}\}$. That is, H is the subgraph obtained by routing all the commodities directly. The maximum out-degree of H, $\Delta^+(H)$, is equal $\max_{s \in \mathcal{S}} |\mathcal{T}(s)|$. Let $s^* = \text{argmax}_{s \in \mathcal{S}} |\mathcal{T}(s)|$. If $s^* = v$, the center vertex, then $W = \{v\}$ and $\mathcal{K}' = \{k \in \mathcal{K} : s_k = v\}$, we obtain the following bound on Δ^*:

$$\Delta^* \geq \text{LB}(W, \mathcal{K}') = \left\lceil \frac{|\mathcal{T}(v)| + 1 - 1}{1} \right\rceil = |\mathcal{T}(v)|,$$

which matches the maximum out-degree of H. Alternatively, $s^* \neq v$. Then when $W = \{s, v\}$ and $\mathcal{K}' = \{k \in \mathcal{K} : s_k = s^*\}$, we obtain the following bound on Δ^*:

$$\Delta^* \geq \text{LB}(W, \mathcal{K}') = \left\lceil \frac{(|\mathcal{T}(s_i)| - 1) + 2 - 1}{2} \right\rceil = \left\lceil \frac{|\mathcal{T}(s_i)|}{2} \right\rceil.$$

Note that we have $|\mathcal{T}(s_i)| - 1$ since it could be that some commodity $k \in \mathcal{K}'$ has source s^* and sink v, which falls within W. Thus, we have $\Delta_H \leq 2\Delta^*$ as desired. □

All of the algorithms presented rely on the lower bound formulation to certify the performance of the algorithm. However, there are instances where the best lower bound of the form provided in Lemma 6.3 is at most $\frac{2}{3}\Delta^*$. Thus, arguing that the performance of an approximation algorithm for the star setting is better than a factor 3/2 cannot rely on the current lower bound construction. Future work is to find hardness results for the star setting, and to improve upon the factor 2 algorithm.

Consider the instance of `min-degree-SPP` in Figure 6.16, where there are $2n$ sinks ($\mathcal{T}$), and $m = \binom{2n}{n}$ sources ($\mathcal{S}$). Consider the m subsets of $\mathcal{T}$ of size n, and order the subsets arbitrarily as $T_1, T_2, \cdots, T_m$. The commodity set is constructed so that each source must route to one of the m subsets of n sinks. That is, $\mathcal{K} = \{(s_i, t) : i \in [m], t \in T_i\}$.

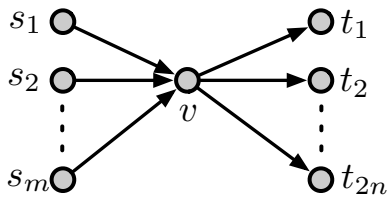


Figure 6.16

Observe that $H = \{s_k t_k : k \in \mathcal{K}\}$ is one solution with $\Delta^+(H) = n$. Furthermore, if in any feasible solution H we had that $\deg^+_H(v) \leq n - 1$, then this would imply that $\deg^+_H(s) \geq n$ for some source s, since v did not serve any of its corresponding sinks. Thus, $\Delta^* = n$. Now consider the family of lower bounds from Lemma 6.3. It is not hard to see that the best lower bound possible is when $W = \{s_i, s_j, v\}$ where $T_i \cap T_j = \emptyset$, and $\mathcal{K}'$ is the set of $2n$ commodities with sources in $\{s_i, s_j\}$. This gives a lower bound of $\lceil (2n + 1)/3 \rceil$.

Theorem 6.21. *There is a polynomial-time 2-approximation for star instances of* `min-degree-SPP`*, and for any $\epsilon > 0$ there is a star instance of* `min-degree-SPP` *where*

$$\max_{W \subseteq N, \mathcal{K}' \subseteq \mathcal{K}} \text{LB}(W, \mathcal{K}') = \frac{2}{3}\Delta^* + \epsilon.$$

6.6.2 Tree instances

A tree instance of `min-degree-SPP` is a *junction tree instance* if there is some node r in D such that r is a node in P_k for all $k \in \mathcal{K}$. Suppose we had an algorithm for junction tree instances of `min-degree-SPP`, $\mathcal{A}$, that returns feasible solutions with max out-degree at most a factor α larger than optimal. In this section, we show how an algorithm for junction tree instances can be used to obtain an approximation algorithm for tree instances.

Let $\mathcal{I} = (D, \mathcal{K})$ be a tree instance of `min-degree-SPP` and let Δ^* denote the minimum max out-degree of a feasible solution. For any node v in D, there is a set of commodities, $\mathcal{K}_v$, such that the dipath between the source and sink in D, P_k, includes v. That is,

$$\mathcal{K}_v := \{k \in \mathcal{K} : v \in P_k\}.$$

For each of the remaining commodities not in $\mathcal{K}_v$, the path P_k is fully contained in one of the components in $D - v$. Naturally, the union of a feasible solution to the instance $(D, \mathcal{K}_v)$ and the instance $(D, \mathcal{K} \setminus \mathcal{K}_v)$ is a feasible solution to the instance $\mathcal{I}$. Moreover, the two subproblems have a particular structure: $(D, \mathcal{K}_v)$ is a junction tree instance, and $(D, \mathcal{K} \setminus \mathcal{K}_v)$ is a set of tree instances – one defined on each component of $D - v$.

Let $v \in N$ and let C be some connected component in $D - v$. Let $\mathcal{K}(C)$ denote the set of commodities $k \in \mathcal{K}$ with $s_k, t_k \in V(C)$. That is,

$$\mathcal{K}(C) := \{k \in \mathcal{K} : s_k, t_k \in V(C)\}.$$

Note that each of the tree instances in $(D, \mathcal{K} \setminus \mathcal{K}_v)$ is the instance $(C, \mathcal{K}(C))$ for some component $C \in D - v$, and C has fewer nodes than D. For each instance $(C, \mathcal{K}(C))$, we again decompose the instance into a junction tree instance and a set of smaller tree instances. We repeat this process until all sub-instances of $\mathcal{I}$ are either junction tree instances, or defined on digraphs with a single node.

Suppose each node in D is in at most β of the junction tree sub-instances of $\mathcal{I}$. Then, applying the algorithm $\mathcal{A}$ to each of the junction tree instances and taking the union of the solutions gives a feasible solution for $\mathcal{I}$, with max out-degree at most $\alpha \cdot \beta$ times Δ^*. It is not hard to show that the node v in each iteration can be chosen so that $\beta = \log(n)$. Specifically, we select the node v so that each connected component in $D - v$ has at most $|N|/2$ nodes.

We define the following algorithm for tree instances.

Algorithm 17: `tree-alg`$(\mathcal{I}, \mathcal{A})$

Input: A tree instance $\mathcal{I} = (D, \mathcal{K})$ of `min-degree-SPP`, where $D = (N, A)$, and an α-approximation algorithm, $\mathcal{A}$, for junction tree instances.

if $|N| = 1$ **then**
  return $(N, \emptyset)$
Let v be a node in N such that each connected component in $D - v$ contains at most $|N|/2$ nodes.
$H_v \leftarrow \mathcal{A}((D, \mathcal{K}_v))$
Let $\mathcal{C} = \{C_1, C_2, \ldots, C_\ell\}$ be the set of connected components in $D - v$.
for $C \in \mathcal{C}$ **do**
  $H_v^C \leftarrow$ `tree-alg`$((C, \mathcal{K}(C))$
$H \leftarrow H_v \bigcup_{C \in \mathcal{C}} H_v^C$
return H

Proposition 6.22. *Let $\mathcal{I} = (D, \mathcal{K})$ be a tree instance of* `min-degree-SPP`*. Given an α-approximation algorithm for junction tree instances, Algorithm 17 returns a solution H for $\mathcal{I}$ with $\Delta^+(H) \leq \alpha \log(n) \Delta^*$*

Proof. We first argue that the solution returned is feasible. Let $k \in \mathcal{K}$. Observe that if $k \notin \mathcal{K}_v$, then P_k must be fully contained in a component C in $D - v$. Thus, at any point in the algorithm when a tree instance is decomposed into a junction tree instance and a set of smaller tree instances, one of the instances contains the path P_k.

Furthermore, it is clear that the minimum max out-degree of any subproblem $\mathcal{I}' = (D', \mathcal{K}')$ generated by the algorithm provides a lower bound on Δ^*, since any solution to $\mathcal{I}$ must also contain a subgraph $H' \subseteq \texttt{cl}(D')$ that is a solution for $\mathcal{I}'$. Finally, each node is contained in at most $\log(n)$ of the subproblems on which algorithm $\mathcal{A}$ is executed. Therefore, $\Delta^+(H) \leq \alpha \log(n) \Delta^*$ as required. □

Thus, we have proven the following Theorem.

Theorem 6.23. *Given a polytime α-approximation algorithm for junction tree instances, there is a polytime $\alpha \log n$-approximation algorithm for tree instances of* `min-degree-SPP`*.*

6.7 Summary and observations

In this chapter, we prove that even in the case where the underlying undirected physical network forms as tree, it is NP-hard to determine whether a feasible sort point allocation exists. We focus on the natural objective of minimizing the maximum number of sort points required at a warehouse.

We present a simple and efficient combinatorial algorithm for solving single-source tree instances of `min-degree-SPP` with a quadratic speed-up over previous algorithms. We also present a fast combinatorial additive 1-approximation algorithm for the out-tree setting. Moreover we prove that our analysis is tight by exhibiting an instance where the algorithm returns a solution that has max out-degree one greater than optimal. It remains open whether or not out-tree instances are NP-hard.

We show that there is an inherent weakness of the family of lower bounds considered since there is a gap of one between the best lower bound and the minimum max out-degree for out-tree instances. For star instances of `min-degree-SPP`, there are instances for which this gap is a multiplicative factor of 3/2. An improvement to the approximation guarantee for the star setting, as well as a non-trivial approximation algorithm for junction tree instances remain open problems. Finally, a challenging open problem lies in finding good approximation algorithms for arbitrary graph structures.

Acknowledgements We thank Sharat Ibrahimpur and Kostya Pashkovich for valuable discussions on lower bounds and the link to matroid intersection.

Chapter 7

Minimum cardinality sort point problem

In contrast to the minimum degree setting, it is not known whether or not the decision version of `min-cardinality-SPP` is NP-hard, even for tree instances. We work towards settling the question of the hardness by presenting exact combinatorial algorithms for various graph structures including in-trees, out-trees, and stars. We then consider merged-tree instances, which are instances defined on graphs obtained by merging an in-tree and out-tree at their root nodes. We prove that merged-tree instances can be reduced to the simpler class of junction tree instances. By taking advantage of uncrossing operations, we show that spider instances, a subclass of merged-tree instances, can be solved efficiently. Additionally, we present a simple 3-approximation algorithm for tree instances.

7.1 Introduction

There are two natural upper bounds on the minimum number of arcs required to route all of the source-sink pairs, denoted m^*. The graph D itself is feasible, as is the arc set formed by including a direct arc between each source-sink pair in the commodity set. Thus, $m^* \leq \min\{|A|, |\mathcal{K}|\}$. As for lower bounds, m^* is at least the size of the source set, $\mathcal{S}$, since each source must have an outbound arc in any feasible solution. Similarly, m^* must be at least the size of the sink set, $\mathcal{T}$.

Fact 7.1. $\max\{|\mathcal{S}|, |\mathcal{T}|\} \leq m^* \leq \min\{|A|, |\mathcal{K}|\}$.

As a result of this bound, we see that in the case of single-source instances and single-sink instances of `min-cardinality-SPP`, an optimal solution is immediate.

Proposition 7.2. *Single-source and single-sink instances of* `min-cardinality-SPP` *can be solved in polynomial time.*

Proof. Suppose there is a unique source, s. The subgraph of $\texttt{cl}(D)$ with arcs $\{st_k : k \in \mathcal{K}\}$ is a feasible solution with $|\mathcal{T}|$ arcs, matching the lower bound. The single-sink setting is analogous. □

7.1.1 Weakly-connected components

When there are multiple sources, we do not know whether or not tree instances can be solved in polynomial time. In progress towards answering this question, we present simple and efficient combinatorial algorithms for special cases including in-tree and out-tree, star, and spider instances. For each of these settings the algorithmic approach relies on splitting the instance into a collection of weakly-connected subproblems based on the *commodity graph*, defined as follows.

Definition 7.3 (commodity graph)**.** *The* commodity graph *for instance $\mathcal{I}$, denoted $G(\mathcal{I})$, is the graph with vertex set $\mathcal{S} \cup \mathcal{T}$ and edge set $\{s_k t_k : k \in \mathcal{K}\}$.*

An example is given in Figure 7.2 for the instance in Figure 7.1.

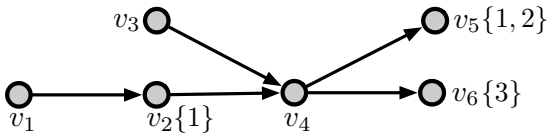


Figure 7.1: Sample tree instance $\mathcal{I}$.

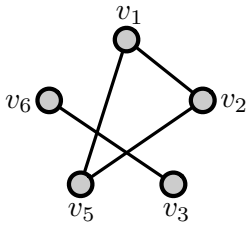


Figure 7.2: Corresponding graph $G(\mathcal{I})$.

Recall that we say an (undirected) graph G is *connected* if for each pair of nodes u and v there is a path between u and v in G. A digraph is *weakly-connected* if its underlying undirected graph is connected. We also define a notion of connectivity for instances of `min-cardinality-SPP`.

Definition 7.4 (weakly-connected)**.** *We say that an instance $\mathcal{I} = (D, \mathcal{K})$ of* `min-cardinality-SPP` *is* weakly-connected *if the corresponding commodity graph $G(\mathcal{I})$ is connected.*

When an instance is weakly-connected, each node that serves as a source or sink must be in the same weakly-connected component in any feasible solution. Thus, we obtain the following improved lower bound when the instance is weakly-connected.

Lemma 7.5. *If $\mathcal{I}$ is weakly-connected then $m^* \geq |\mathcal{S} \cup \mathcal{T}| - 1$.*

An instance $\mathcal{I}$ of `min-cardinality-SPP` can be partitioned into a set of weakly-connected subinstances as follows. Suppose the commodity graph has ℓ maximally-connected components. Let N_i denote the node set of the ith component, for each $i \in [\ell]$. The ith subinstance of $\mathcal{I}$, denoted $\mathcal{I}_i = (D, \mathcal{K}_i)$, is the `min-cardinality-SPP` instance with base graph D and commodity set $\mathcal{K}_i = \{k \in \mathcal{K} : s_k, t_k \in N_i\}$. We refer to $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell\}$ as the *partition of $\mathcal{I}$ into weakly-connected subproblems*. Note that $N_i = \mathcal{S}_i \cup \mathcal{T}_i$, where $\mathcal{S}_i$ and $\mathcal{T}_i$ denote the set of sources and sinks for the commodity set $\mathcal{K}_i$. For each $i \in [\ell]$, let m_i^* denote the minimum number of arcs in a feasible subgraph for instance $\mathcal{I}_i$. Observe that the following lemma holds, since $\bigcup_{i\in[\ell]} \mathcal{K}_i = \mathcal{K}$, and in any feasible solution for $\mathcal{I}$, each subset of nodes N_i must be in the same weakly-connected component.

Lemma 7.6. $\sum_{i\in[\ell]}(|N_i| - 1) \leq m^* \leq \sum_{i\in[\ell]} m_i^*$.

In general, the union of optimal solutions to each weakly-connected subinstance is not an optimal solution for the original instance. In particular, there are instances for which $m^* < \sum_{i\in[\ell]} m_i^*$. One such example is given in Figure 7.3. The node sets of the connected components of $G(\mathcal{I})$ are $N_1 = \{v_1, v_2, v_3, v_9, v_{11}, v_{12}\}$ and $N_2 = \{v_4, v_5, v_6, v_8, v_{13}, v_{14}\}$.

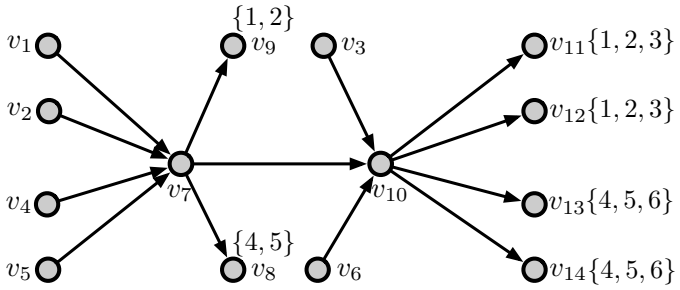


Figure 7.3

Optimal solutions to instances $\mathcal{I}_1$ and $\mathcal{I}_2$ are given in Figures 7.4 and 7.5, respectively, each with 7 arcs. By combining the subgraphs we obtain a solution for $\mathcal{I}$ with 14 arcs. However an optimal solution for $\mathcal{I}$, the digraph D itself, has only 13 arcs.

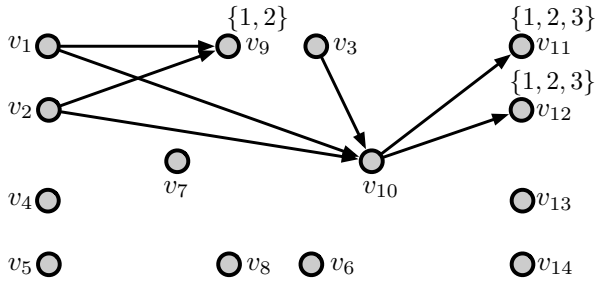


Figure 7.4

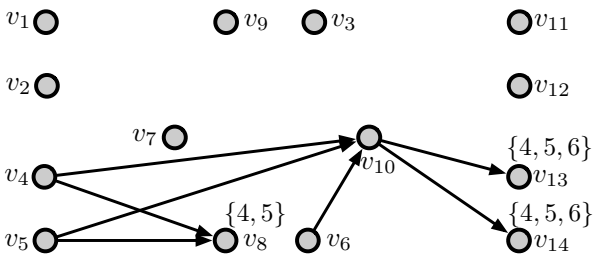


Figure 7.5

In Section 7.5 we prove that for many classes of instances, it is indeed the case that $m^* = \sum_{i\in[\ell]} m_i^*$.

7.1.2 Formulation

The algorithms presented in Section 7.5 are combinatorial in nature, since the linear relaxation of the natural formulation is non-integral. Let $\mathcal{C} = \{C \subseteq N : \exists k \in \mathcal{K} \text{ such that } s_k \in C, t_k \notin C\}$. Since we focus on tree instances, any s_k, t_k-dipath in $\texttt{cl}(D)$ is a dipath in $\texttt{cl}(P_k)$ by Lemma 1.1. As a result, we obtain the following formulation for tree instances of `min-cardinality-SPP`.

$$\begin{array}{lll} \min & x(\texttt{cl}(D)) & \\ \text{s.t.} & x(\delta^+_{cl(D)}(C)) \geq 1, & \forall C \in \mathcal{C} \\ & x \in \{0,1\} & \end{array} \qquad \text{(IP-MCSPP)}$$

The linear relaxation for this formulation is not integral, even for path instances. Consider the following example with 2 sources and 2 sinks, where one source and one sink are internal nodes (non-leaves). There are three commodities with source-sinks $(v_1, v_3), (v_1, v_4)$, and (v_2, v_3), where the commodity paths are the unique dipath given in Figure 7.6. While any integer solution requires at least 3 arcs, we see in Figure 7.7 that there is a fractional solution with value 2.5. We can use the same class of instances to obtain a gap that converges to 4/3, as proven in the following theorem.

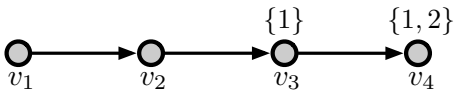


Figure 7.6: Base graph

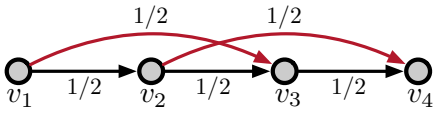


Figure 7.7: Fractional solution

Theorem 7.7. *Formulation (IP-MCSPP) has an integrality gap of at least* $4/3-\epsilon$ *for any* $\epsilon > 0$ *on path instances.*

Proof. Let ℓ be a positive integer. Let $D = (N, A)$ be the path given on the ordered set of nodes $\{v_1, v_2, \ldots, v_{2\ell}\}$ in Figure 7.8. Let $\texttt{cl}(D) = (N, E)$. Let $\mathcal{K}$ be the set of commodities with source-sink pairs $\bigcup_{j\in[\ell]}\{(v_i, v_{j+\ell}) : i \in [j]\}$, and uniquely determined designated paths. Observe that each node in D is the source or sink node of some commodity in $\mathcal{K}$ and the instance $D, \mathcal{K}$ is weakly-connected. Thus, any feasible solution has at least $2\ell - 1 = \frac{4\ell-2}{2}$ arcs.

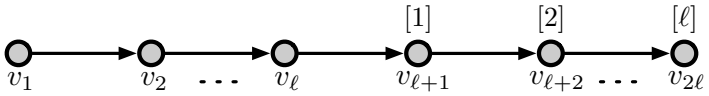


Figure 7.8: Path instance with 2ℓ nodes.

Let $\bar{x} \in \mathbb{R}^E$ be the vector such that $\bar{x}_{v_iv_{i+1}} = 0.5$ for all $i \in [2\ell - 1]$, and $\bar{x}_{v_iv_{\ell+i}} = 0.5$ for all $i \in [\ell]$, and on all other arcs the value is zero. It follows that cost of $\bar{x}$ is $\frac{3\ell-1}{2}$. We claim that $\bar{x}$ is feasible for (IP-MCSPP).

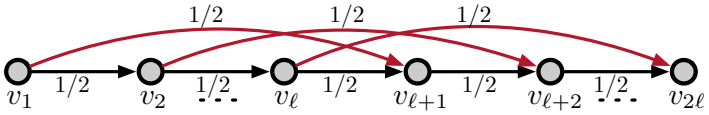


Figure 7.9: Fractional solution.

Let $C \in \mathcal{C}$, and let $k \in \mathcal{K}$ such that $s_k \in C$ and $t_k \notin \mathcal{C}$. It follows that $s_k = v_i$ and $t_k = v_{\ell+j}$ for some $j \geq i$. Let $W = \{s_k = v_i, v_{i+1}, \ldots, v_q\}$ be the maximal set of contiguous nodes in W and P_k, starting at v_i. If there is a node $v_p \in C \cap W$ with $q + 1 < p < \ell + j$, assume p is the largest index of such a node. Then $\bar{x}(\delta^+_{cl(D)}(C)) \geq \bar{x}_{v_q,v_{q+1}} + \bar{x}_{v_p,v_{p+1}} \geq 1$ as required.

Alternatively, there is no such node v_p. If $q < \ell+i$, then $\bar{x}(\delta^+_{cl(D)}(C)) \geq \bar{x}_{v_q,v_{q+1}}+\bar{x}_{v_i,v_{\ell+i}} \geq 1$. Otherwise, $q \geq \ell + i$. As a result, $v_j \in W$, and so $\bar{x}(\delta^+_{cl(D)}(C)) \geq \bar{x}_{v_q,v_{q+1}} + \bar{x}_{v_j,v_{\ell+j}} \geq 1$.

Therefore, the integrality gap is at least $\frac{4\ell-2}{3\ell-1}$. For any $\epsilon > 0$, setting $\ell > \frac{3\epsilon+2}{9\epsilon}$ gives the desired result. □

7.2 Reduction techniques

In this section we introduce a procedure to reduce the size of tree instances by removing *Steiner nodes*.

Definition 7.8 (Steiner node)**.** *Let $\mathcal{I} = (D, \mathcal{K})$ be an instance of* `min-cardinality-SPP`*, with $D = (N, A)$. A node $v \in N$ is a* Steiner node *if $v \notin \mathcal{S} \cup \mathcal{T}$.*

Let $\mathcal{W}$ denote the set of Steiner nodes. If a Steiner node v has in-degree or out-degree at most one in the graph D, then we will prove that there is an optimal solution that does not include v. We will use this procedure to generate a 3-approximation for tree instances in Section 7.3) and as a sub-process for the exact algorithms in Section 7.5. We use two methods to remove these Steiner nodes and reduce instance sizes: node deletion and arc contraction.

Definition 7.9 (node deletion)**.** *Let $\mathcal{I} = (D, \mathcal{K})$ and let $v \in N$ such that $v \notin P_k \ \forall k \in \mathcal{K}$. The instance obtained by deleting node v, is $\mathcal{I} \setminus v := (D \setminus v, \mathcal{K})$, where $D \setminus v$ denotes the graph obtained from D after deleting node v.*

Definition 7.10 (edge contraction)**.** *Let $\mathcal{I} = (D, \mathcal{K})$ and let $e \in A$. Then the instance obtained by contracting arc $e = uv$, is $\mathcal{I}/e := (D/e, \mathcal{K}/e)$, where D/e denotes the graph obtained by contracting e in D, and $\mathcal{K}/e$ denotes the commodity set obtained by contracting e in any commodity path in $\mathcal{K}$, and replacing any source or sink in $\{u, v\}$ with the new merged node.*

By iteratively applying these two reduction operations we can remove all Steiner nodes with in-degree or out-degree at most one in D. This procedure is stated in the following algorithm. Note that in each iteration, the instance $\mathcal{I}$ along with the digraph D changes, and so the set of Steiner nodes with in-degree or out-degree at most one must also be updated in each iteration.

Algorithm 18: `reduce`$(\mathcal{I})$

Input: Instance $\mathcal{I} = (D, \mathcal{K})$ of `min-cardinality-SPP`.

$\bar{\mathcal{W}} \leftarrow \{v \in \mathcal{W} : \max\{\deg_D^+(v), \deg_D^-(v)\} \leq 1\}$
while $\bar{\mathcal{W}} \neq \emptyset$ **do**
  Let v be in $\bar{\mathcal{W}}$
  if $\min\{\deg_D^+(v), \deg_D^-(v)\} = 0$ **then**
    $\mathcal{I} \leftarrow \mathcal{I} \setminus v$
  if $\deg_D^+(v) = 1$ **then**
    $e = vw \leftarrow \delta_D^+(v)$
    $\mathcal{I} \leftarrow \mathcal{I}/e$, labelling the merged node as w
  else
    $e = uv \leftarrow \delta_D^-(v)$
    $\mathcal{I} \leftarrow \mathcal{I}/e$, labelling the merged node as u
  Update $\bar{\mathcal{W}}$
return $\mathcal{I}$

A sample execution of the algorithm is given in Figures 7.10 - 7.12. Figure 7.10 provides an instance which has Steiner node set $\{v_4, v_5, v_6\}$. Since v_5 has in-degree zero, it is removed, resulting in the instance in Figure 7.11. Steiner node v_4 has out-degree one, so arc v_4v_6 is contracted, and the resulting instance is given in Figure 7.12. At this point the set $\bar{\mathcal{W}}$ is empty, since the only remaining Steiner node has in-degree 3 and out-degree 2.

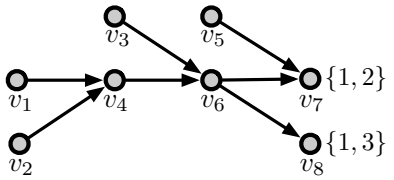


Figure 7.10: Original instance.

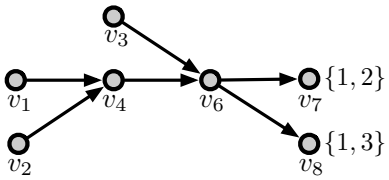


Figure 7.11: v_5 removed.

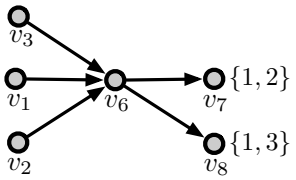


Figure 7.12: v_4v_6 contracted.

Lemma 7.11. *Let $\mathcal{I}'$ be the instance returned by Algorithm 18. Any feasible solution for $\mathcal{I}'$ is feasible for $\mathcal{I}$. Moreover, any optimal solution for $\mathcal{I}'$ is optimal for $\mathcal{I}$.*

Proof. Let H' be a feasible solution for instance $\mathcal{I}'$, and let $k \in \mathcal{K}$. Observe that the two actions, node deletion and edge contraction retain the nodes s_k and t_k, as well as maintain a path P_k' in D' that is an s_k, t_k subpath of `cl`(P_k). Since H' is feasible for $\mathcal{I}'$, there is

an s_k, t_k-dipath in $H' \cap \texttt{cl}(P'_k)$, and so the same dipath exists in $H' \cap cl(D)$. Therefore H' is feasible for instance $\mathcal{I}$. We now argue that an optimal solution for $\mathcal{I}'$ will indeed be optimal for $\mathcal{I}$.

Let H be a feasible subgraph of $\texttt{cl}(D)$ for instance $\mathcal{I}$. We will show that H can be converted to a feasible solution to $\mathcal{I}'$ with no additional arcs. Let $\{v_1, v_2, \ldots, v_t\}$ be the ordered set of Steiner nodes considered by Algorithm 18, and let D_i be the digraph obtained after removing the ith Steiner node via node deletion or arc contraction. Similarly, let $\mathcal{I}_i$ be the resulting `min-cardinality-SPP` instance after removing the ith Steiner node. We proceed by induction on t. If $t = 0$, the the result trivially holds. Suppose the statement holds when $t = \ell$ for some $\ell \geq 0$, and consider the case where $t = \ell + 1$.

By induction, there is a subgraph H_{t-1} that is feasible for $\mathcal{I}_{t-1}$ with at most the number of arcs as H. If $\deg^+_{D_{t-1}}(v_t) = 0$ or $\deg^-_{D_{t-1}}(v_t) = 0$, then $H_t = H_{t-1} \setminus v_t$ is feasible for $\mathcal{I}_t$. Otherwise, suppose $\deg^+_{D_{t-1}}(v_t) = 1$ and let e_t be the unique arc in $\delta^+_{D_{t-1}}(v_t)$. Then $H_t = H_{t-1}/e_t$ is feasible for $\mathcal{I}_t$. An analogous argument holds when $\deg^-_{D_i}(v_i) = 1$. Since each operation cannot increase the size of the subgraph, H_t is feasible for $\mathcal{I}' = \mathcal{I}_t$ with size at most that of H as required. □

Even if a Steiner node has in-degree and out-degree at least two, it may not be used in any optimal solution. In fact, perhaps surprisingly, even for tree instances it can be the case that every optimal solution contains an (undirected) cycle and does not include Steiner nodes. One such example is presented in Figure 7.13, with Steiner nodes v_4 and v_6. Let H be a feasible solution. If both v_4 and v_6 are non-singleton in H, then $H = D$ and has 7 arcs. If only v_4 is non-singleton in H, then again H must have at least 7 arcs as shown in Figure 7.14. The case where only v_6 is in H is analogous. The only optimal solution is given in Figure 7.15, which contains a cycle and no Steiner nodes.

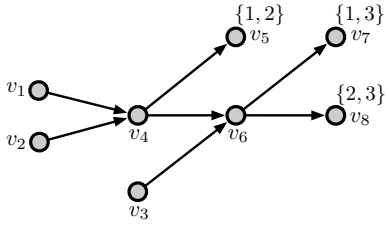


Figure 7.13

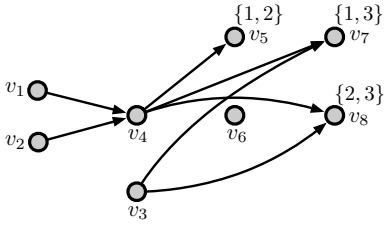


Figure 7.14

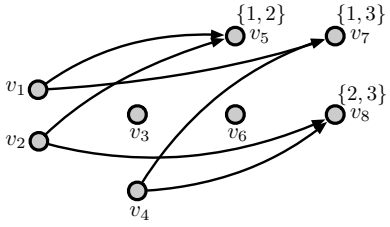


Figure 7.15

7.3 3-approximation for tree instances

As a result of Section 7.2, we may assume that all Steiner nodes in tree instances have in-degree and out-degree at least two in D. To obtain an approximation algorithm, we bound the number of Steiner nodes that can be present in such an instance.

Lemma 7.12. *Let $\mathcal{I}$ be a tree instance of* `min-cardinality-SPP`*. If each Steiner node has in-degree and out-degree at least two, then $|\mathcal{W}| \leq \frac{|\mathcal{S}\cup\mathcal{T}|}{2} - 1$.*

Proof. Let G be the underlying undirected graph of D, with n vertices and $n-1$ edges. Let n_1 denote the number of source and sink nodes that are leaves in G, and let n_2 denote the number of sources and sink nodes that are not leaves in G. By definition, $n_1+n_2 = |\mathcal{S}\cup\mathcal{T}|$, and $n = n_1 + n_2 + |\mathcal{W}|$. Since the total degree of G is $2(n-1)$ and each Steiner node has degree at least four, we obtain the following inequality.

$$n_1 + 2n_2 + 4|\mathcal{W}| \leq 2(n_1 + n_2 + |\mathcal{W}| - 1).$$

Rearranging gives

$$2|\mathcal{W}| \leq n_1 - 2 \leq |\mathcal{S} \cup \mathcal{T}| - 2,$$

and so $|\mathcal{W}| \leq \frac{|\mathcal{S}\cup\mathcal{T}|}{2} - 1$. □

Since the base graph itself is feasible for any instance, we obtain the following result.

Theorem 7.13. *There is a polytime 3-approximation algorithm for tree instances of* `min-cardinality-SPP`*.*

Proof. Let $\mathcal{I}' = (D', \mathcal{K}) =$ `reduce`$(\mathcal{I})$. By construction, $\mathcal{S}' \cup \mathcal{T}' = \mathcal{S} \cup \mathcal{T}$.

Let D' be the base graph of $\mathcal{I}'$. By Lemma 7.12, $|\mathcal{W}'| \leq \frac{|\mathcal{S}'\cup\mathcal{T}'|}{2} - 1 = \frac{|\mathcal{S}\cup\mathcal{T}|}{2} - 1$. Thus, D' is a feasible solution with at most

$$|\mathcal{S} \cup \mathcal{T}| + |\mathcal{W}'| - 1 \leq |\mathcal{S} \cup \mathcal{T}| + \frac{|\mathcal{S} \cup \mathcal{T}|}{2} - 2 \leq \frac{3}{2}|\mathcal{S} \cup \mathcal{T}| - 2$$

arcs. Since $m^* \geq \max\{|\mathcal{S}|, |\mathcal{T}|\}$ and $|\mathcal{S} \cup \mathcal{T}| \leq 2\max\{|\mathcal{S}|, |\mathcal{T}|\} \leq 2m^*$, it follows that D' has at most $3m^*$ arcs, as required. □

We obtain the following corollary by replacing the bound $m^* \geq \max\{|\mathcal{S}|, |\mathcal{T}|\}$ with the bound $m^* \geq |\mathcal{S} \cup \mathcal{T}| - 1$.

Corollary 7.14. *There is a $\frac{3}{2}$-approximation algorithm for weakly-connected tree instances of* `min-cardinality-SPP`.

Proof. By the same argument as above, the graph $D' = (N', A')$ is feasible and has at most $\frac{3}{2}|\mathcal{S} \cup \mathcal{T}| - 2$ arcs. Since $\mathcal{I}$ is weakly-connected, $m^* \geq |\mathcal{S} \cup \mathcal{T}| - 1$. Thus, $|A'| \leq \frac{3}{2}m^*$. □

7.4 Uncrossing

We have discussed decomposing an instance into its weakly-connected subinstances, and simplifying an instance by removing Steiner nodes of low degree. The final technique central to the analysis and execution of the exact algorithms presented in Section 7.5 is that of *uncrossing*.

Definition 7.15 (Crossing)**.** *Let $\mathcal{I} = (D, \mathcal{K})$ be a tree instance of* `min-cardinality-SPP`. *A pair of arcs v_1w_1 and v_2w_2 in $\texttt{cl}(D)$ are* crossing *if the unique v_1, w_1- and v_2, w_2-dipaths in D share at least one arc.*

Figures 7.16 to 7.18 show three examples of crossing arcs, where the straight arcs are those that form D.

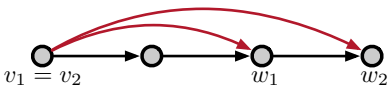


Figure 7.16

Figure 7.17

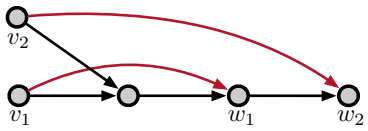


Figure 7.18

The following uncrossing operations can be applied to a feasible solution $H \subseteq \texttt{cl}(D)$ to reduce the number of crossing arcs while maintaining feasibility.

Definition 7.16 (Uncrossing operations)**.** *Let $H \subseteq \texttt{cl}(D)$. We refer to the following operations as* uncrossing operations.

1. *If H contains arcs $v_1w \neq v_2w$ where v_2 is on the unique path in D from v_1 to w, replace v_1w with v_1v_2.*
2. *If H contains arcs $vw_1 \neq vw_2$ where w_1 is on the unique path in D from v to w_2, replace vw_2 with w_1w_2.*

Operation 1 is shown in the transition from Figure 7.19 to 7.20, and operation 2 is shown in the transition from Figure 7.21 to 7.22.

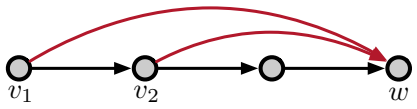


Figure 7.19

Figure 7.20

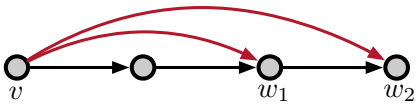


Figure 7.21

Figure 7.22

Observe that applying the uncrossing operations cannot increase the number of arcs in a solution, and maintains feasibility.

Definition 7.17 (Minimally-crossing)**.** *Let $\mathcal{I} = (D, \mathcal{K})$ be a tree instance and let $H \subseteq \texttt{cl}(D)$. We say that H is* minimally-crossing *if it is not possible to apply the uncrossing operations to any pair of arcs.*

Lemma 7.18 (Uncrossing Lemma I)**.** *Let H be a minimally-crossing subgraph of $\texttt{cl}(D)$. Then for any pair of nodes v_1, v_2, if there is a node w such that*

- *H contains both a v_1, w-dipath and a v_2, w-dipath, and*
- *v_2 is on the unique v_1, w-dipath in D,*

then H contains a v_1, v_2-dipath.

Proof. For a contradiction, suppose H does not contain a v_1, v_2-dipath for such a pair v_1 and v_2. Let w be the node closest to v_1 and v_2 in D such that H has a v_i, w-dipath for each $i \in [2]$. Let Q_i denote a fixed v_i, w-dipath in H for each $i \in [2]$ and let $e_1 = u_1 w$ and $e_2 = u_2 w$ be the final arcs in Q_1 and Q_2 respectively.

By choice of w, it follows that $u_1 \neq u_2$. Without loss of generality, u_2 is on the unique dipath from u_1 to w since $\mathcal{I}$ is a tree instance. Therefore, uncrossing operation 1 can be applied to the arcs $u_1 w$ and $u_2 w$, a contradiction. □

The following lemma is analogous, as is its proof; rather than applying uncrossing operation 1 we apply operation 2.

Lemma 7.19 (Uncrossing Lemma II)**.** *Let H be a minimally-crossing subgraph of* $\texttt{cl}(D)$*. Then for any pair of nodes v_1, v_2, if there is a node w such that*

- *H contains both a w, v_1-dipath and a w, v_2-dipath, and*
- *v_1 is on the unique w, v_2-dipath in D,*

then H contains a v_1, v_2-dipath.

Rather than solving an instance directly, it is often convenient to instead work with an *equivalent instance under uncrossing.*

Definition 7.20. *Let $\mathcal{I}_1 = (D, \mathcal{K}_1)$ and $\mathcal{I}_2 = (D, \mathcal{K}_2)$ be two instances of* `min-cardinality-SPP` *defined on the same base graph. We say that $\mathcal{I}_1$ and $\mathcal{I}_2$ are* equivalent under uncrossing *if any minimally-crossing subgraph $H \subseteq \texttt{cl}(D)$ is feasible for $\mathcal{I}_1$ if and only if it is feasible for $\mathcal{I}_2$.*

Uncrossing Lemmas I and II imply the following lemma.

Lemma 7.21. *Let $\mathcal{I} = (D, \mathcal{K})$ be tree instance of* `min-cardinality-SPP` *and suppose v and v' are a pair of nodes in D. If D contains a v, v'-dipath and*

$$(\mathcal{S}(v) \cup \mathcal{T}(v)) \cap (\mathcal{S}(v') \cup \mathcal{T}(v')) \neq \emptyset,$$

then instances $\mathcal{I}$ and $\mathcal{I}' = (D, \mathcal{K} \cup \{(v, v')\})$ are equivalent under uncrossing.

Proof. Certainly any feasible solution for $\mathcal{I}'$ is feasible for $\mathcal{I}$. Let H be a minimally-crossing solution to $\mathcal{I}$. We claim that H must contain a v, v'-dipath if D contains a v, v'-dipath and there is a node $w \in (\mathcal{S}(v) \cup \mathcal{T}(v)) \cap (\mathcal{S}(v') \cup \mathcal{T}(v'))$.

Since D contains a v, v'-dipath and $\mathcal{I}$ is a tree instance, $\mathcal{S}(v) \cap \mathcal{T}(v') = \emptyset$. Therefore, there exists a node w such that either $w \in \mathcal{S}(v) \cap \mathcal{S}(v')$, $w \in \mathcal{T}(v) \cap \mathcal{T}(v')$, or $w \in \mathcal{T}(v) \cap \mathcal{S}(v')$. If $w \in \mathcal{S}(v) \cap \mathcal{S}(v')$, then Uncrossing Lemma II implies that H contains a v, v'-dipath. Similarly, if $w \in \mathcal{T}(v) \cap \mathcal{T}(v')$, Uncrossing Lemma I implies that H contains a v, v'-dipath. Finally, if $w \in \mathcal{T}(v) \cap \mathcal{S}(v')$, then H contains a v, w-dipath and a w, v'-dipath, which forms a v, v'-dipath. Therefore $\mathcal{I}'$ and $\mathcal{I}$ are equivalent under uncrossing as required. □

The following algorithm takes as input a tree instance $\mathcal{I}$, and repeatedly adds commodity pairs that satisfy the conditions of Lemma 7.21. Note that the sets $\mathcal{S}(v)$ and $\mathcal{T}(v)$ are updated in each iteration for all $v \in N$ since $\mathcal{K}$ gains commodities.

Algorithm 19: `Augment`$(\mathcal{I})$

Input: $\mathcal{I} = (D, \mathcal{K})$, a tree instance.

Let E be the arc set of `cl`(D)

while $\exists vv' \in E$ *such that* $(\mathcal{S}(v) \cup \mathcal{T}(v)) \cap (\mathcal{S}(v') \cup \mathcal{T}(v')) \neq \emptyset$ *and* $(v, v') \notin \mathcal{K}$ **do**

  $\mathcal{K} \leftarrow \mathcal{K} \cup \{(v, v')\}$

return $\mathcal{I}$

Since equivalence under uncrossing is transitive, we obtain the following corollary.

Lemma 7.22. *Let $\mathcal{I} = (D, \mathcal{K})$ be a tree instance and let $\mathcal{I}^{max} = (D, \mathcal{K}^{max}) :=$* `Augment`$(\mathcal{I})$*. Then $\mathcal{I}$ and $\mathcal{I}^{max}$ are equivalent under uncrossing.*

Since $\mathcal{I}$ and $\mathcal{I}^{max}$ are equivalent under uncrossing, and $\mathcal{K} \subseteq \mathcal{K}^{max}$, given an instance $\mathcal{I}$ it suffices to solve $\mathcal{I}^{max}$. We call an instance $\mathcal{I}$ *maximal* if $\mathcal{I}$ and $\mathcal{I}^{max}$ have the same commodity sets. In Section 7.5.4 we present an algorithm for spider instances that relies on working with maximal instances.

7.5 Exact algorithms

In this section we first present exact combinatorial algorithms for solving in-tree, out-tree, and star instances of `min-cardinality-SPP`. We define a merged-tree as the union of an in-tree, T^{in}, and an out-tree, T^{out}, merged at the root nodes, r. We prove that merged-tree instances can be reduced to junction tree instances. We then build upon this result to prove that spider instances of `min-cardinality-SPP` can be solved in polynomial time. In this section, we will assume all Steiner nodes with in- or out-degree at most one in D have been removed.

7.5.1 In-tree and out-tree instances

Recall that $D = (N, A)$ is an in-tree (out-tree) if for each node $v \in N$, $\deg_D^+(v) \leq 1$ ($\deg_D^-(v) \leq 1$). Since each Steiner node must have in-degree and out-degree at least two after preprocessing, we may assume such instances have no Steiner nodes. It follows that if $\mathcal{I}$ is weakly-connected, D itself is an optimal solution since it has $|\mathcal{S} \cup \mathcal{T}| - 1$ arcs, matching the lower bound in Lemma 7.5. By the same argument, each weakly-connected subinstance $\mathcal{I}_i$ of $\mathcal{I}$ has a solution with $|\mathcal{S}_i \cup \mathcal{T}_i| - 1$ arcs. As a result, the union of optimal solutions to the weakly-connected subproblems is in fact an optimal solution for $\mathcal{I}$. Algorithm 20

returns an optimal solution for any in-tree or out-tree instance of `min-cardinality-SPP` in polynomial time.

Algorithm 20: `in-out-tree`$(\mathcal{I})$

Input: Instance $\mathcal{I} = (D, \mathcal{K})$ of `min-cardinality-SPP`, where $D = (N, A)$ is an in-tree or out-tree.

1 $H \leftarrow (N, \emptyset)$
2 Let $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell\}$ be the partition of $\mathcal{I}$ into weakly-connected subinstances
3 **for** $i \in \{1, 2, \ldots, \ell\}$ **do**
4 $\mathcal{I}'_i = (D'_i, \mathcal{K}'_i) \leftarrow$ `reduce`$(\mathcal{I}_i)$
5 $H \leftarrow H \cup D'_i$
6 **return** H

A sample execution of the algorithm is given in Figures 7.23 to 7.33. The input instance is given in Figure 7.23, and the optimal solution obtained by Algorithm 20 is given in Figure 7.24.

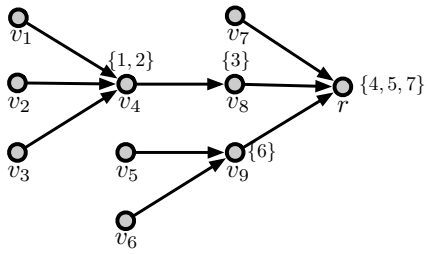


Figure 7.23: In-tree instance.

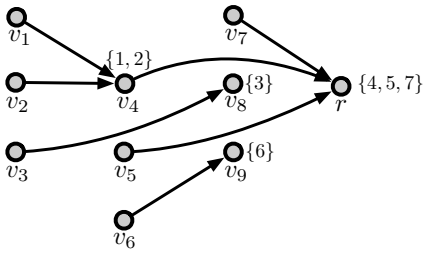


Figure 7.24: Optimal solution.

The partition of $\mathcal{I}$ into weakly-connected subinstances gives the instances $\mathcal{I}_1, \mathcal{I}_2$ and $\mathcal{I}_3$ in Figures 7.25 - 7.27.

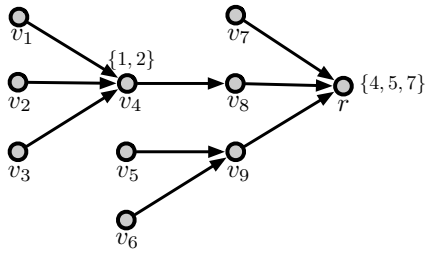


Figure 7.25: Instance $\mathcal{I}_1$.

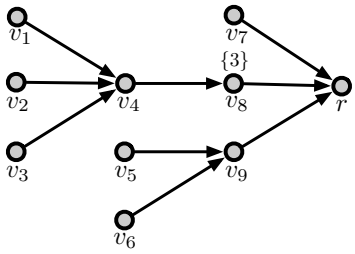


Figure 7.26: Instance $\mathcal{I}_2$.

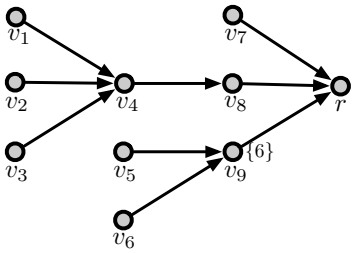


Figure 7.27: Instance $\mathcal{I}_3$.

The contracted graphs (optimal solutions) are given in Figures 7.28 - 7.30, with the corresponding arcs shown on the full node set of D shown in Figures 7.31 - 7.33.

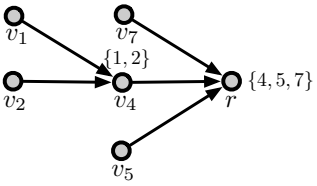


Figure 7.28: Digraph D'_1.

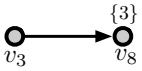


Figure 7.29: Digraph D'_2.

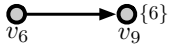


Figure 7.30: Digraph D'_3.

Combining the three subgraphs of $\texttt{cl}(D)$ in Figures 7.31 - 7.33 gives the optimal solution in Figure 7.24.

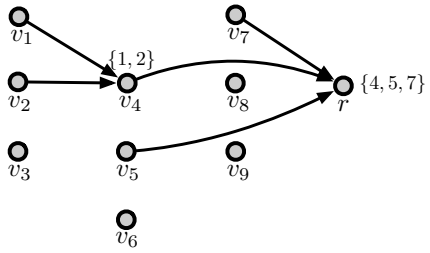


Figure 7.31:
Arc set of D'_1 on N.

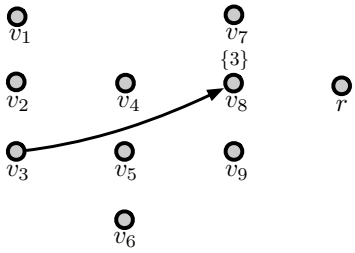


Figure 7.32:
Arc set of D'_2 on N.

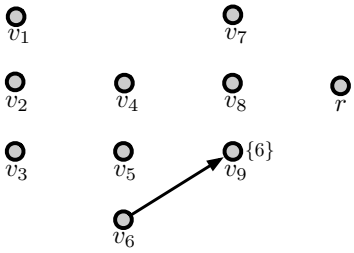


Figure 7.33:
Arc set of D'_3 on N.

Theorem 7.23. *In-tree and out-tree instances of* `min-cardinality-SPP` *can be solved in polynomial time.*

Proof. We prove that Algorithm 20 returns an optimal solution for `min-cardinality-SPP` given an in- or out-tree instance. By Lemma 7.11, every feasible solution to $\texttt{reduce}(\mathcal{I})$

is also feasible for $\mathcal{I}$. Let $i \in [\ell]$. Since D'_i is feasible for instance $\mathcal{I}'_i$, it is also feasible for instance $\mathcal{I}_i$. As a result, $H = \bigcup_{i\in[\ell]} D'_i$ is feasible for instance $\mathcal{I}$. Additionally, Algorithm 18 returns an in-tree (out-tree) instance when it is given an in-tree (out-tree) instance as input. Therefore for each $i \in [\ell]$, no Steiner node with respect to instance $\mathcal{I}_i$ remains in D'_i, and so the number of arcs in D'_i is equal to $|N_i| - 1$. Therefore, $|H| = \sum_{i\in[\ell]}(|N_i| - 1)$, and so H is optimal since its arc count matches the lower bound stated in Lemma 7.6. □

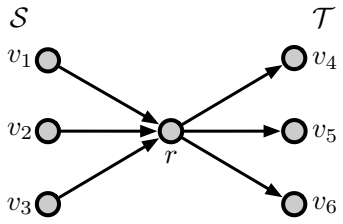
We obtain the following corollary.

Corollary 7.24. *When $\mathcal{I}$ is an in-tree or out-tree instance, $m^* = \sum_{i\in[\ell]} m^*_i$.*

Additionally, we see that path instances of `min-cardinality-SPP` can be solved in polynomial time, even though the natural formulation has an integrality gap of at least $4/3-\epsilon$ for all $\epsilon > 0$ for path instances.

Corollary 7.25. *When D is a directed path,* `min-cardinality-SPP` *can be solved in polynomial time.*

7.5.2 Star instances



Let $\mathcal{I} = (D, \mathcal{K})$ be a star instance, and let r denote the unique non-leaf node in $D = (N, A)$. Due to preprocessing, we may assume each node $v \in N \setminus r$ with out-degree (in-degree) one in D is a source (sink). It follows that each arc $vw \in \texttt{cl}(D)$ has either $v \in \mathcal{S}$ or $w \in \mathcal{T}$ (or both). Intuitively, this observation suggests that each arc in $\texttt{cl}(D)$ is "useful" only to commodities that share a source or sink with an endpoint of the arc. This motivates the following result.

Lemma 7.26. *When $\mathcal{I}$ is a star instance $m^* = \sum_{i\in[\ell]} m^*_i$.*

Proof. Let $H = (N, A')$ be an optimal solution for instance $\mathcal{I}$. Let $\{N_1, N_2, \ldots, N_\ell\}$ denote the node sets of the maximally-connected components in $G(\mathcal{I})$, and let $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell\}$ be the partition of $\mathcal{I}$ into weakly-connected subinstances.

Let $i \in [\ell]$, and let H_i be the subgraph of H which includes all arcs with an endpoint in $N_i \setminus \{r\}$. That is, $H_i = (N, A_i)$ where $A_i := \{uv \in A' : \{u, v\} \cap (N_i \setminus \{r\}) \neq \emptyset\}$. For each commodity $k \in \mathcal{K}_i$, the s_k, t_k-dipath in H consists of at most two arcs, each with at least one endpoint in $\{s_k, t_k\}$ which is not equal to r. Therefore H_i is feasible for instance $\mathcal{I}_i$.

Moreover, for each $j \neq i$, if an arc uv is in A_i and A_j, then without loss of generality, u is a source node in $\mathcal{I}_i$ and v is sink node in $\mathcal{I}_j$, each not equal to r. However, such an arc could be removed while maintaining feasibility. Thus since H is an optimal solution, no such arc exists and $A_i \cap A_j = \emptyset$. As a result, $m^* \geq \sum_{i \in [\ell]} m_i^*$, and by Lemma 7.6 we have that $m^* = \sum_{i \in [\ell]} m_i^*$. □

Therefore, an optimal solution for $\mathcal{I}$ is the union of optimal solutions for each weakly-connected subinstance, and it suffices to solve star instances that are weakly-connected. Since there is at most one Steiner node, the node r, it follows that an optimal solution to a weakly-connected star instance has at most $|N| - 1 \leq |\mathcal{S} \cup \mathcal{T}|$ arcs (the base graph itself), and at least $|\mathcal{S} \cup \mathcal{T}| - 1$ arcs. If the base graph is not optimal, then we will show that the arc set $\{s_k t_k : k \in \mathcal{K}\}$ forms an optimal solution.

Lemma 7.27. *Let $\mathcal{I} = (D, \mathcal{K})$ be a weakly-connected star instance. If D is not an optimal solution, then $\{s_k t_k : k \in \mathcal{K}\}$ forms an optimal solution.*

Proof. Let $H = (N, A')$ be an optimal solution for instance $\mathcal{I}$. For each commodity $k \in \mathcal{K}$, the only two possible s_k, t_k-dipaths in D are s_k, r, t_k or s_k, t_k. Since D is not an optimal solution, r must be a singleton in H, as otherwise H has at least $|N| - 1$ arcs, the same as D. Moreover, $r \notin \mathcal{S} \cup \mathcal{T}$ and $A' \subseteq \{st : s \in \mathcal{S}, t \in \mathcal{T}\}$. If there is an arc $e \in A' \setminus \{s_k t_k : k \in \mathcal{K}\}$, then e can be removed from H while maintaining feasibility, contradicting that H was optimal. Therefore, $A' = \{s_k t_k : k \in \mathcal{K}\}$ as required. □

By combining Lemmas 7.26 and 7.27, we see that the following Algorithm returns an optimal solution given a star instance.

Algorithm 21: `star`$(\mathcal{I})$

Input: A star instance $\mathcal{I} = (D, \mathcal{K})$ of `min-cardinality-SPP`, with center node r and graph $D = (N, A)$

$A' \leftarrow \emptyset$
$\{N_1, N_2, \ldots, N_\ell\} \leftarrow$ node sets of the maximal connected components in $G(\mathcal{I})$
$\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell\} \leftarrow$ partition of $\mathcal{I}$ into weakly-connected sub-instances
for $i \in [\ell]$ **do**
    if $|N_i \cup r| - 1 \leq |\mathcal{K}_i|$ **then**
        $A' \leftarrow A' \cup \{s_k r : k \in \mathcal{K}_i\} \cup \{r t_k : k \in \mathcal{K}_i\}$
    else
        $A' \leftarrow A' \cup \{s_k t_k : k \in \mathcal{K}_i\}$
return $H = (N, A')$

Theorem 7.28. *Star instances of* `min-cardinality-SPP` *can be solved in polynomial time.*

7.5.3 Merging in- and out-trees

A digraph D is a *merged-tree* if it is obtained by merging an in-tree, $T^{in} = (N^{in}, A^{in})$, and an out-tree, $T^{out} = (N^{out}, A^{out})$, at their root nodes. Let r denote the merged node in D, which we refer to as the *join*. A junction tree instance of `min-cardinality-SPP` is a merged-tree instance where the unique commodity path P_k includes the node r for each $k \in \mathcal{K}$. In this section, we prove that the task of finding a polytime algorithm for merged-tree instances reduces to the task of finding a polytime algorithm for junction tree instances.

The distinction between merged-tree and junction tree instances is depicted in Figures 7.34 and 7.35. In a merged-tree instance, the source and sink sets can contain any node in D, whereas in a junction tree instance, the sources and sinks must be contained in the in-tree and out-tree, respectively.

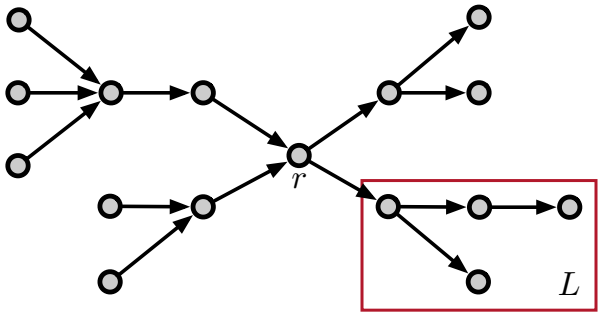


Figure 7.34: Merged-tree.

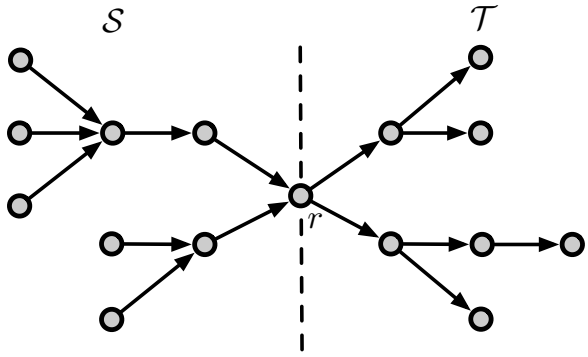


Figure 7.35: Junction tree instance.

We refer to maximally-connected subgraphs of D that do not contain the node r as *legs* of the merged-tree, such as L indicated in Figure 7.34. Let $\mathcal{L}$ denote the set of legs of the merged-tree.

The set of commodities can be decomposed into commodities whose source and sink are on the same leg, denoted $\mathcal{K}_{\mathcal{L}}$, and commodities whose source and sink are on different legs, or have an endpoint equal to r, denoted $\mathcal{K}_{\mathcal{J}}$. Equivalently,

$$\mathcal{K}_{\mathcal{L}} := \{k \in \mathcal{K} : r \notin P_k\} \text{ and } \mathcal{K}_{\mathcal{J}} := \{k \in \mathcal{K} : r \in P_k\}.$$

If a merged-tree instance has $\mathcal{K}_{\mathcal{L}} = \emptyset$, then it is in fact a junction tree instance. Given instance $\mathcal{I}$, we define the subinstances $\mathcal{I}_{\mathcal{L}}$ and $\mathcal{I}_{\mathcal{J}}$ as the instances on the same digraph with commodity sets $\mathcal{K}_{\mathcal{L}}$ and $\mathcal{K}_{\mathcal{J}}$ respectively. That is, $\mathcal{I}_{\mathcal{L}} := (D, \mathcal{K}_{\mathcal{L}})$ and $\mathcal{I}_{\mathcal{J}} := (D, \mathcal{K}_{\mathcal{J}})$. Observe that $\mathcal{I}_{\mathcal{L}}$ is the union of disjoint in-tree and out-tree instances, and $\mathcal{I}_{\mathcal{J}}$ is a junction tree instance. For example, given the instance $\mathcal{I}$ in Figure 7.36, the instances $\mathcal{I}_{\mathcal{L}}$ and $\mathcal{I}_{\mathcal{J}}$ are given in Figures 7.37 and 7.38.

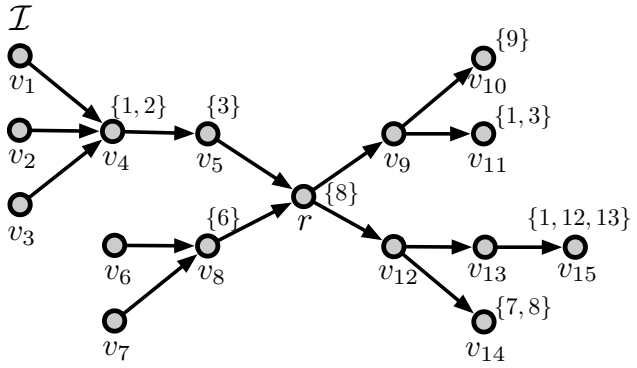


Figure 7.36: Merged-tree instance $\mathcal{I}$

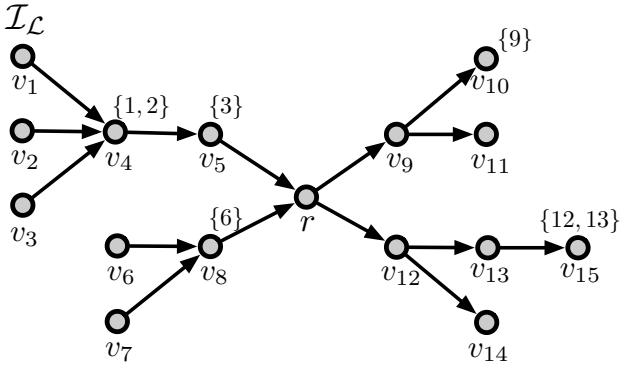


Figure 7.37: $\mathcal{I}_{\mathcal{L}}$

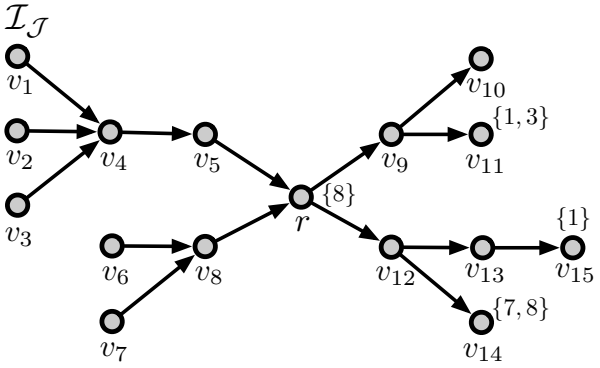


Figure 7.38: $\mathcal{I}_{\mathcal{J}}$

We will prove that in order to solve a merged-tree instance $\mathcal{I} = (D, \mathcal{K})$, it suffices to find an optimal minimally-crossing solution to $\mathcal{I}_{\mathcal{L}}$, and an optimal solution to a junction tree instance obtained from $\mathcal{I}_{\mathcal{J}}$ by contracting arcs that are included in the minimally-crossing solution to $\mathcal{I}_{\mathcal{L}}$. We first show that in any minimally-crossing solution to $\mathcal{I}_{\mathcal{L}}$, all nodes in the same weakly-connected component have a path to (or from) the node in the same component that is "closest" to r in D. We define a natural measurement of distance as follows.

For any pair of nodes u and v, let $d(u, v)$ be equal to the number of arcs in the unique path between u and v in D. Let $W \subseteq N$. The *representative node for W in D*, denoted $\texttt{rep}_D(W)$ is the node in W that is closest to r in D, breaking ties arbitrarily. That is,

$$\texttt{rep}_D(W) = \text{argmin}_{w \in W} d(w, r).$$

Let $\mathcal{N} = \{r, N_1, N_2, \ldots, N_\ell\}$ be the node sets of the maximally-connected components in

$G(\mathcal{I}_{\mathcal{L}})$, along with the singleton r. Let $R = \{r, r_1, r_2, \ldots, r_\ell\}$ denote the corresponding set of representative nodes. For the instance $\mathcal{I}_{\mathcal{L}}$ given in Figure 7.37,

$$\mathcal{N} = \{r, \{v_1, v_2, v_4\}, \{v_3, v_5\}, \{v_6, v_8\}, \{v_7\}, \{v_9, v_{10}\}, \{v_{11}\}, \{v_{12}, v_{13}, v_{15}\}, \{v_{14}\}\}, \text{ and}$$

$$R = \{r, v_4, v_5, v_8, v_7, v_9, v_{11}, v_{12}, v_{14}\}.$$

We will prove that in any minimally-crossing solution H for $\mathcal{I}_{\mathcal{L}}$, each node has a path in H to (or from) its representative node. First, we prove that this result holds in D. That is, we prove that in an in-tree (out-tree) instance, each node has a dipath in D to (from) the representative node in the same weakly-connected component. Then, we prove that when H is a minimally-crossing subgraph of $\texttt{cl}(D)$, if v and w are in the same weakly-connected component in H and D contains a v, w-dipath, then H contains a v, w-dipath. We first prove the following more general result regarding dipaths in D.

Lemma 7.29. *Let D be in-tree (out-tree) and let $H \subseteq \texttt{cl}(D)$. Let W be the node set of a weakly-connected component in H and let $r' = \texttt{rep}_D(W)$. For each node $v \in W$, D contains a v, r'-dipath (r', v-dipath).*

Proof. We prove the result only for in-trees, since the out-tree case is analogous. For a contradiction, suppose D does not contain a v, r'-dipath. Then certainly r' is not equal to the root node, r. By definition, D contains a unique v, r-dipath, P_v, and a unique r, r'-dipath, $P_{r'}$. Since there is no v, r'-dipath, then r' is not in P_v and by definition of r', v is not in $P_{r'}$. Since D is an in-tree, P_v and $P_{r'}$ share a maximal dipath P_w, which is a path from a node w to r, as depicted in Figure 7.39.

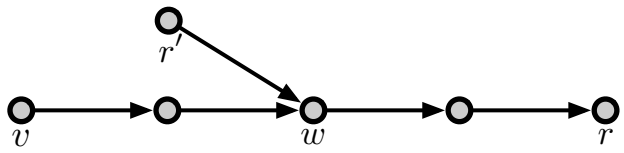


Figure 7.39

Let $C_{r'}$ and C_v be the components in $D \setminus w$ containing r' and v respectively. Since r' and v are in the same weakly-connected component in H, there is some sequence of arcs connecting the two nodes. However, the only arcs connecting the nodes in $C_{r'}$ and the nodes in C_v to the rest of the graph must have an endpoint on the path P_w. Thus, N' must have a node along the path P_w, contradicting the designation of r' as a representative node, since $d(w, r) < d(r', r)$. Therefore D contains a v, r'-dipath. □

For each $i \in [\ell]$, let L_i be the leg in $\mathcal{L}$ that contains the node set N_i. Note that we may have $L_i = L_j$ for $i \neq j$.

Corollary 7.30. *For each $i \in [\ell]$,*

- *if $N_i \subseteq T^{in}$, L_i contains a v, r_i-dipath for each $v \in N_i$, and*
- *if $N_i \subseteq T^{out}$, L_i contains an r_i, v-dipath for each $v \in N_i$.*

Proof. Let $i \in [\ell]$, and suppose $N_i \subseteq N^{in}$. Let $H_i := \texttt{cl}(L_i)[N_i]$. That is, H_i is the subgraph of $\texttt{cl}(L_i)$ consisting of all arcs with both endpoints in the set N_i. By definition of N_i, H_i is weakly-connected, and $r_i = \texttt{rep}_D(N_i)$. By Lemma 7.29 and since L_i is an in-tree, it follows that L_i contains a v, r_i-dipath for each $v \in N_i$. The case where $N_i \subseteq N^{out}$ is analogous. □

For in-tree and out-tree instances, we now prove that in any minimally-crossing solution H, if two nodes v and w are in the same weakly-connected component in H and D contains a v, w-dipath, then H must also contain a v, w-dipath.

Lemma 7.31. *Let D be an in-tree or out-tree and let H be a minimally-crossing subgraph of $\texttt{cl}(D)$. If v and w are in the same weakly-connected component in H and D contains a v, w-dipath, then H contains a v, w-dipath.*

Proof. Suppose D is an in-tree, and that v and w are nodes in the same weakly-connected component in H. Furthermore, suppose D contains a v, w-dipath. Since v and w are in the same weakly-connected component, there is some minimal sequence of arcs $e_1, e_2, \ldots, e_p$ in H where v is an endpoint of e_1, w is an endpoint of e_p, and e_i and e_{i+1} share an endpoint for each $i \in [p-1]$. Let $e_i = v_i w_i$ for each $i \in [p]$. We proceed by induction on p. If $p = 1$, then $v_1 = v$ and $w_1 = w$, and the result follows. Suppose the statement holds for all commodity sequences of length less than $p = \alpha$ for some $\alpha \geq 1$, and consider a sequence of length $p \geq \alpha + 1$.

Suppose $v = v_1$. Then D has both a v, w_1-dipath and a v, w-dipath. Since the sequence is minimal, w_1 is equal to v_2 or w_2, and so w_1 and w are connected by a sequence of arcs of length $p - 1$. Since each node has out-degree one in D, either w is on the unique v, w_1-dipath in D, or w_1 is on the unique v, w-dipath in D. If w is on the unique v, w_1-dipath in D, then by induction, there is a w, w_1-dipath in H. Since H also contains a v, w_1-dipath Uncrossing Lemma I (Lemma 7.18) implies that H has a v, w-dipath. If instead w_1 is on the unique v, w-dipath in D, then by induction H contains a w_1, w-dipath. Concatenating the v, w_1- and w_1, w-dipaths gives a v, w-dipath in D.

Otherwise, suppose $v = w_1$. Then D contains a v_1, w-dipath since it has a v_1, w_1-dipath and a w_1, w-dipath. Moreover, since D is an in-tree, v is on the unique dipath in D between v_1 and w. By minimality, v_1 is equal to v_2 or w_2 and so by induction, H contains a v_1, w-dipath. By Uncrossing Lemma II (Lemma 7.19), H contains a v, w-dipath.

The case where D is an out-tree follows by an analogous argument, where instead of considering the first arc in the sequence, we consider the final arc in the sequence. □

Note that this lemma is not true for general digraphs, and relies on the fact that D is an in-tree or out-tree. Consider the instance $\mathcal{I} = (D, \mathcal{K})$ given in Figure 7.40, as well as the minimally-crossing solution H given in Figure 7.41. Despite the fact that nodes v_2 and v_4 are in the same weakly-connected component in H and that D contains a v_2, v_4-dipath, H does not contain a v_2, v_4-dipath.

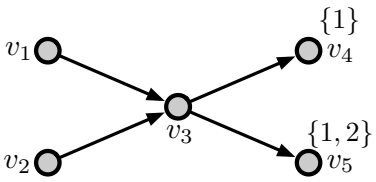


Figure 7.40

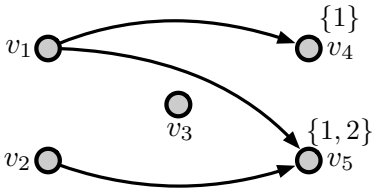


Figure 7.41

Combining Corollary 7.30 and Lemma 7.31 allows us to prove the following result.

Lemma 7.32. *Let H be a minimally-crossing solution for $\mathcal{I}_{\mathcal{L}}$. For each $i \in [\ell]$,*

- *if $N_i \subseteq T^{in}$, H contains a v, r_i-dipath for each $v \in N_i$, and*
- *if $N_i \subseteq T^{out}$, H contains an r_i, v-dipath for each $v \in N_i$.*

Proof. Let H be a minimally-crossing solution to $\mathcal{I}_{\mathcal{L}}$. Let $i \in [\ell]$, and suppose $N_i \subseteq N^{in}$. By definition, the node set N_i is contained in the some weakly-connected component of $H \cap \texttt{cl}(L_i)$, a minimally-crossing subgraph of L_i. By Corollary 7.30, L_i contains a v, r_i-dipath for each $v \in N_i$. By Lemma 7.31, it follows that H contains a v, r_i-dipath for each $v \in N_i$. The case where $N_i \subseteq N^{out}$ is analogous. □

We construct a new junction tree instance by shifting all sources and sinks to their corresponding representative node. Let $\phi_{\mathcal{N},R} : N \to R$ denote the function that maps each node to its representative node. That is, for each $v \in N$, if $v \in N_i$, then $\phi_{\mathcal{N},R}(v) = r_i$. As a minor abuse of notation, let $\phi_{\mathcal{N},R}(\mathcal{K}) = \{(\phi_{\mathcal{N},R}(s_k), \phi_{\mathcal{N},R}(t_k)) : k \in \mathcal{K}\}$.

Definition 7.33. *Let $\mathcal{I}_\mathcal{J} = (D, \mathcal{K}_\mathcal{J})$ be a junction tree instance of* `min-cardinality-SPP` *with $D = (N, A)$. Let $\mathcal{N} = \{r, N_1, N_2, \ldots, N_\ell\}$ be a partition of N with corresponding representative nodes $R = \{r, r_1, r_2, \ldots, r_\ell\}$ in D. The instance obtained by* merging $\mathcal{I}_\mathcal{J}$ according to partition $\mathcal{N}$ *is $\phi_\mathcal{N}(\mathcal{I}_\mathcal{J}) := (D, \phi_{\mathcal{N},R}(\mathcal{K}_\mathcal{J}))$.*

For example, given the partition $\mathcal{N}$ and representative nodes from instance $\mathcal{I}_\mathcal{L}$ in Figure 7.37, $\mathcal{N} = \{r, \{v_1, v_2, v_4\}, \{v_3, v_5\}, \{v_6, v_8\}, \{v_7\}, \{v_9, v_{10}\}, \{v_{11}\}, \{v_{12}, v_{13}, v_{15}\}, \{v_{14}\}\}$, and $R = \{r, v_4, v_5, v_8, v_7, v_9, v_{11}, v_{12}, v_{14}\}$, and junction tree instance $\mathcal{I}_\mathcal{J}$ in Figure 7.42, we obtain $\phi_\mathcal{N}(\mathcal{I}_\mathcal{J})$ as shown in Figure 7.43.

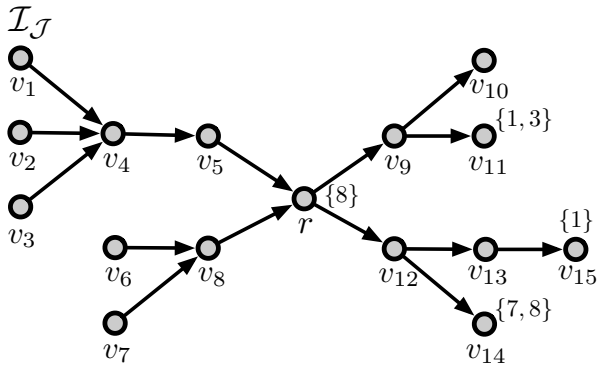


Figure 7.42: $\mathcal{I}_\mathcal{J}$

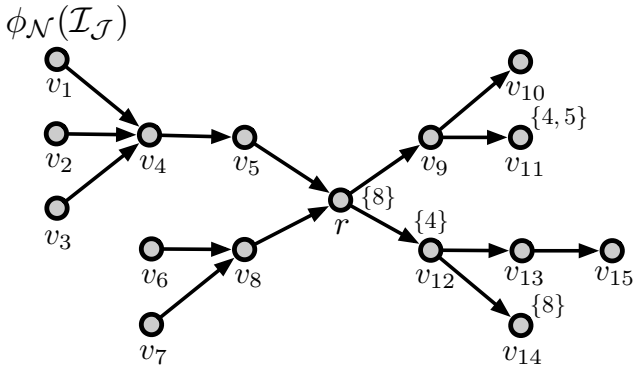


Figure 7.43: $\phi_\mathcal{N}(\mathcal{I}_\mathcal{J})$

When $\mathcal{N}$ is the partitioning of N into the node sets of the maximally-connected components in $G(\mathcal{I}_\mathcal{L})$, it follows that $\phi_\mathcal{N}(\mathcal{I}_\mathcal{J})$ is also a junction tree instance since each node in N^{in} is mapped to a representative node in N^{in}, and each node in N^{out} is mapped to representative node in N^{out}. We now state the main result in this subsection: solving the merged-tree instance $\mathcal{I}$ reduces to the problem of solving $\mathcal{I}_\mathcal{L}$ and $\phi_\mathcal{N}(\mathcal{I}_\mathcal{J})$.

Theorem 7.34. *Let $\mathcal{I} = (D, \mathcal{K})$ be a merged-tree instance of* `min-cardinality-SPP` *with join r. Let $\mathcal{I}_\mathcal{L} = (D, \mathcal{K}_\mathcal{L})$ and $\mathcal{I}_\mathcal{J} = (D, \mathcal{K}_\mathcal{J})$ where*

$$\mathcal{K}_\mathcal{L} = \{k \in \mathcal{K} : r \notin P_k\} \text{ and } \mathcal{K}_\mathcal{J} = \{k \in \mathcal{K} : r \in P_k\}.$$

Let $\mathcal{N} = \{N_1, N_2, \ldots, N_\ell\}$ be the node sets of the maximally-connected components in $G(\mathcal{I}_\mathcal{L})$. If H_1 and H_2 are optimal solutions for $\mathcal{I}_\mathcal{L}$, and $\phi_\mathcal{N}(\mathcal{I}_\mathcal{J})$ respectively, and H_1 is minimally-crossing, then $H_1 \cup H_2$ is an optimal solution for $\mathcal{I}$.

First, we prove the following lemma.

Lemma 7.35. *$H_1 \cup H_2$ is feasible for $\mathcal{I}$.*

Proof. Since $\mathcal{I}_{\mathcal{L}}$ and $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$ have the same base graph as instance $\mathcal{I}$, we know that $H = H_1 \cup H_2 \subseteq \texttt{cl}(D)$. Thus, it suffices to show that for each commodity $k \in \mathcal{K}$, H contains an s_k, t_k-dipath. If $k \in \mathcal{K}_{\mathcal{L}}$, then by definition of instance $\mathcal{I}_{\mathcal{L}}$ and subgraph H_1, there is an s_k, t_k-dipath in $H_1 \subseteq H$.

Otherwise, $k \in \mathcal{K}_{\mathcal{J}}$. Let N_1 and N_2 denote the node sets in $\mathcal{N}$ containing s_k and t_k respectively, and let r_1 and r_2 denote the corresponding representative nodes. By construction, $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$ has a commodity with source r_1 and sink r_2, and so H_2 must contain an r_1, r_2-dipath. By Lemma 7.32, H_1 contains an s_k, r_1-dipath and an r_2, t_k-dipath. Concatenating the three dipaths gives an s_k, t_k-dipath in H. □

It remains to prove that $H_1 \cup H_2$ is in fact *optimal*. Let $H \subseteq \texttt{cl}(D)$. For each $L \in \mathcal{L}$ let $H[L]$ be the subgraph of H induced by the nodes in L. That is, $H[L] = H \cap \texttt{cl}(L)$. Let $H|_{\mathcal{L}}$ be the union of the subgraphs $H[L]$ over all $L \in \mathcal{L}$, and let $H|_{\mathcal{J}} = H \setminus H|_{\mathcal{L}}$. Note that $H|_{\mathcal{L}}$ consists of all arcs in H with both endpoints in the same leg, and $H|_{\mathcal{J}}$ consists of all arcs in H with either both endpoints in different legs, or an endpoint equal to r.

We will show that any feasible solution can be modified to decompose into optimal solutions to $\mathcal{I}_{\mathcal{L}}$ and $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. One of the main considerations is in the fact that an optimal solution may have sets N_i and N_j in the same weakly-connected component in $H|_{\mathcal{L}}$. In the following lemma, the third condition is not necessary for proving Theorem 7.34. However, it allows us to further restrict the set of junction tree instances that must be considered, and will be used when solving spider instances in Section 7.5.4.

Lemma 7.36. *Let $\mathcal{I} = (D, \mathcal{K})$ be a merged-tree instance of* `min-cardinality-SPP` *with join r. Let H_1 be a minimally-crossing optimal solution to $\mathcal{I}_{\mathcal{L}}$. There is an optimal solution H for $\mathcal{I}$, where*

1. *$H_1 \subseteq H$,*
2. *$H_2 := H \setminus H_1$ is an optimal solution to $\phi_{\mathcal{N}}(I_{\mathcal{J}})$, and*
3. *$H|_{\mathcal{J}} \subseteq cl(D)[V]$.*

where V is the set of representative nodes of the maximally-connected components in $H|_{\mathcal{L}}$.

Proof. Let H be a minimally-crossing optimal solution to instance $\mathcal{I}$, and let H_1 be a minimally-crossing optimal solution to instance $\mathcal{I}_{\mathcal{L}}$. Let $\mathcal{N} = \{r, N_1, N_2, \ldots, N_\ell\}$ be the the node sets of the maximally-connected components in $G(\mathcal{I}_{\mathcal{L}})$, along with the singleton r. Let $\mathcal{V} = \{r, V_1, V_2, \ldots, V_p\}$ be the coarsening of the partition $\mathcal{N}$ obtained by merging node sets N_i and N_j if they are connected in $H|_{\mathcal{L}}$. That is, $\mathcal{V}$ is the partition of N into node sets of the maximally-connected components in $H|_{\mathcal{L}}$, along with the singleton r. Let

$V = \{r, v_1, v_2, \ldots, v_p\} \subseteq R$ denote the corresponding representative nodes. An example instance is given in Figure 7.34, and an example minimally-crossing solution H is given in Figure 7.45, where the solid arcs form $H|_{\mathcal{L}}$, and the dashed arcs form $H|_{\mathcal{J}}$. The sets $\mathcal{N}, R, \mathcal{V}$, and V are

$$\begin{aligned}
\mathcal{N} &= \{r, \{v_1, v_2, v_4\}, \{v_3, v_5\}, \{v_6, v_8\}, \{v_7\}, \{v_9, v_{10}\}, \{v_{11}\}, \{v_{12}, v_{13}, v_{15}\}, \{v_{14}\}\}, \\
R &= \{r, v_4, v_5, v_8, v_7, v_9, v_{11}, v_{12}, v_{14}\}, \\
\mathcal{V} &= \{r, \{v_1, v_2, v_3, v_4, v_5\}, \{v_6, v_7, v_8\}, \{v_9, v_{10}\}, \{v_{11}\}, \{v_{12}, v_{13}, v_{15}\}, \{v_{14}\}\}, \text{ and} \\
V &= \{r, v_5, v_8, v_9, v_{11}, v_{12}, v_{14}\}.
\end{aligned}$$

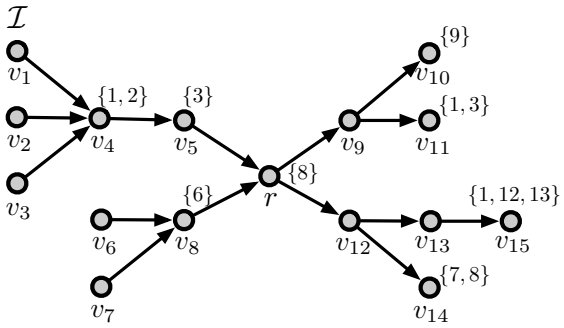


Figure 7.44: Merged-tree instance $\mathcal{I}$.

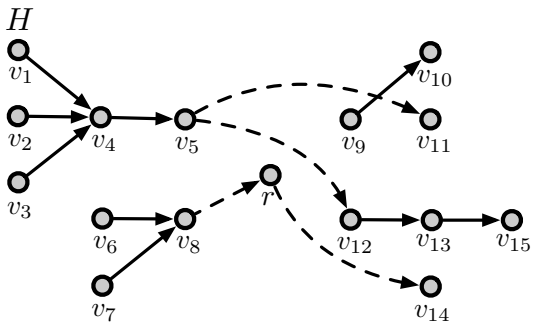


Figure 7.45: Minimally-crossing solution H

Let $i \in [p]$ and let L_i denote the leg containing V_i. By Lemma 7.29, if $V_i \subseteq N^{in}(N^{out})$, then L_i has a v, v_i-dipath (v_i, v-dipath) for each $v \in V_i$. By Lemma 7.31, it follows that if $V_i \subseteq N^{in}(N^{out})$, then $H|_{\mathcal{L}}$ has a v, v_i-dipath (v_i, v-dipath) for each $v \in V_i$.

Claim 1. $H|_{\mathcal{J}} \subseteq \texttt{cl}(D)[V]$.

For a contradiction, suppose there is an arc uw in $H|_{\mathcal{J}}$ where $u \in V_i$ and $u \neq v_i$. We previously established that H contains a u, v_i-dipath. Let ux be the first arc on this path. Since u and x are on the same leg, and u and w are on different legs, it follows that x is on the unique u, w-dipath in D. However, then Uncrossing operation 1 can be applied to arcs ux and uw, contradicting that H is minimally crossing. The case where $w \in V_i$ and $w \neq v_i$ is analogous. This establishes Claim 1.

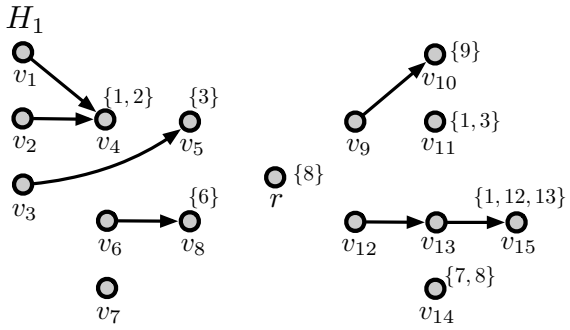


Figure 7.46: H_1

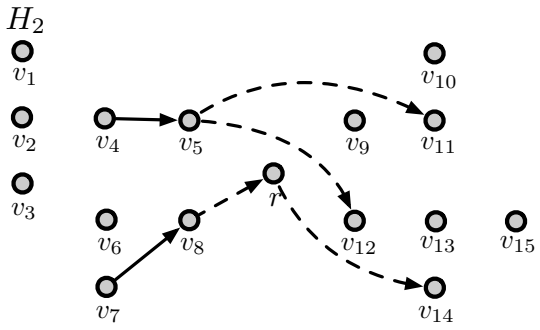


Figure 7.47: H_2

Let G be the digraph on N obtained by including an arc connecting each representative node in N to its representative node in V. That is, G has arc set

$$\bigcup_{i\in[p]:V_i\subseteq N^{in}} \{r_j v_i : j \in [\ell], N_j \subseteq V_i\} \cup \bigcup_{i\in[p]:j\in[\ell],V_i\subseteq N^{out}} \{v_i r_j : N_j \subseteq V_i\}.$$

Since p of these arcs will be loops, after removing loops G contains $\ell - p$ arcs.

Let $H_2 = G \cup H|_{\mathcal{J}}$. The subgraph H_1 is shown in Figure 7.46, and H_2 is given in Figure 7.47. The solid arcs are those in G, and the dashed arcs are those in $H|_{\mathcal{J}}$. Let $H' = H_1 \cup H_2$. We will prove that H' is the desired subgraph. By construction, the node sets of the maximal connected components in $H'|_{\mathcal{L}}$ and $H|_{\mathcal{L}}$ are the same. By Claim 1 and since $H'|_{\mathcal{J}} = H|_{\mathcal{J}}$, it follows that $H'|_{\mathcal{J}} \subseteq \mathtt{cl}(D)[V]$.

It remains to show that H' is an optimal solution to $\mathcal{I}$, and that H_2 is feasible for $\phi_{\mathcal{N}}(I_{\mathcal{J}})$. Note that by Lemma 7.35, since H_1 is feasible for $\mathcal{I}_{\mathcal{L}}$, in order to prove that H' is feasible for $\mathcal{I}$, it suffices to prove that H_2 is feasible for $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. In addition, we will show that $|H'| = |H_1| + |H_2| \leq |H|$, where the cardinality of a subgraph is equal to its arc count.

First, we claim that $H_2 = G \cup H|_{\mathcal{J}}$ is feasible for $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. Let $k \in \phi_{\mathcal{N}}(\mathcal{K}_{\mathcal{J}})$, with source r_i and sink r_j, where r_i and r_j are the representative nodes for a pair of sets N_i and N_j in $\mathcal{N}$. Moreover, by definition of $\phi_{\mathcal{N}}(\mathcal{K}_{\mathcal{J}})$, there is some commodity $k \in \mathcal{K}_{\mathcal{J}}$ such that $s_k \in N_i$ and $t_k \in N_j$, where $N_i \subseteq N^{in}$ and $N_j \subseteq N^{out}$.

Let L_1 be the leg that contains s_k, and let L_2 be the leg that contains t_k. By feasibility of H, there is a dipath from s_k to t_k in H, that must contain some arc uv where $u \in L_1$ and $v \in L_2$, or arcs ur and rv where $u \in L_1$ and $v \in L_2$. Note that by definition of $\mathcal{V}$, $\{s_k, r_i, u\} \subseteq V'$ and $\{v, r_j, t_k\} \subseteq V''$ for some $V', V'' \in \mathcal{V}$. Let v' and v'' be the representative nodes of V' and V'' respectively.

Observe that G contains the (possibly trivial) arcs $r_i v'$ and $v'' r_j$. Additionally, $H|_{\mathcal{J}}$ contains the arc $v'v''$ or the arcs $v'r$ and rv''. In either case, we obtain an r_i, r_j-dipath in H_2 as required.

It remains to prove that $|H_1| + |G| + |H|_{\mathcal{J}}| \leq |H|$. Recall that H decomposes into $H|_{\mathcal{L}}$ and $H|_{\mathcal{J}}$, and that $H_2 = G \cup H|_{\mathcal{J}}$. Thus, it suffices to show that $|H_1| + |G| \leq |H|_{\mathcal{L}}|$. By Corollary 7.24, $|H_1| = \sum_{i \in [\ell]}(|N_i| - 1) = \sum_{i \in [\ell]} |N_i| - \ell$. Additionally, $|G| = \ell - p$. By definition of the partition $\mathcal{V}$ and since all nodes in a single part in $\mathcal{N}$ must be connected in $H|_{\mathcal{L}}$, it follows that $|H|_{\mathcal{L}}| \geq \sum_{i \in [p]}(|V_j| - 1) = \sum_{i \in [\ell]} |N_i| - p$. Thus, $|H_1| + |G| \leq |H|_{\mathcal{L}}|$, as required. □

Proof of Theorem 7.34. Let H_1 and H_2 be optimal solutions for $\mathcal{I}_{\mathcal{L}}$, and $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$ respectively, and suppose H_1 is minimally-crossing. By Lemma 7.35, $H_1 \cup H_2$ is feasible for $\mathcal{I}$. By Lemma 7.36 it is in fact an optimal solution since there is an optimal solution for instance $\mathcal{I}$ that decomposes into optimal solutions for $\mathcal{I}_{\mathcal{L}}$ and $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. □

7.5.4 Spider instances

The main take-away from Section 7.5.3 is that in order to solve a merged-tree instance, it suffices to solve a union of disjoint in-tree and out-tree instances, $\mathcal{I}_{\mathcal{L}}$, and a junction tree instance $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. We have presented an algorithm for solving in-tree and out-tree instances, so it remains to find an algorithm for junction tree instances. For spider instances, a special case of merged-trees, we present an efficient algorithm that solves $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. One of the key considerations is to work with only *maximal* spider instances, since these permit additional structure on $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. An example of the base graph for a spider instance is given in Figure 7.48.

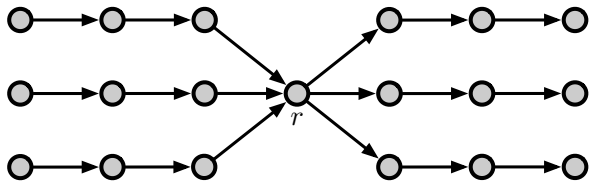


Figure 7.48

We will present an algorithm for solving a special case of spider instances, defined as follows.

Definition 7.37. *An instance $\mathcal{I} = (D, \mathcal{K})$ is a* simple spider instance *if it is both a spider and junction tree instance, and satisfies the following conditions:*
(K1) for all distinct $s, s' \in \mathcal{S} \setminus \{r\}$, $ss' \in \texttt{cl}(D) \Rightarrow \mathcal{T}(s) \cap \mathcal{T}(s') = \emptyset$;
(K2) for all distinct $t, t' \in \mathcal{T} \setminus \{r\}$, $tt' \in \texttt{cl}(D) \Rightarrow \mathcal{S}(t) \cap \mathcal{S}(t') = \emptyset$.

Recall that a tree instance $\mathcal{I} = (D, \mathcal{K})$ is *maximal* if for all pairs of nodes v, v' for which D contains a v, v'-dipath and $(\mathcal{S}(v) \cup \mathcal{T}(v)) \cap (\mathcal{S}(v') \cup \mathcal{T}(v')) \neq \emptyset$, it follows that $(v, v') \in \mathcal{K}$. Moreover, given a tree instance $\mathcal{I}$, $\mathcal{I}^{max} = \texttt{Augment}(\mathcal{I})$ (Algorithm 19) is a maximal instance that equivalent under uncrossing to $\mathcal{I}$. We will prove that any maximal spider instance decomposes into instances $\mathcal{I}_{\mathcal{L}}$ and $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$, where $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$ is a simple spider instance. Then we will present a polytime algorithm that solves simple spider instances.

As a preliminary result, we first show that the following commodities must be in any maximal merged-tree instance.

Lemma 7.38. *Let $\mathcal{I} = (D, \mathcal{K})$ be a maximal merged-tree instance. Let $\mathcal{N} = \{N_1, N_2, \ldots, N_\ell\}$ be the node sets of the maximally-connected components in $G(\mathcal{I}_{\mathcal{L}})$ and let $R = \{r_1, r_2, \ldots, r_\ell\}$ be the corresponding set of representative nodes in D. For each $i \in [\ell]$,*

1. *if $N_i \subseteq N^{in}$, $(v, r_i) \in \mathcal{K}$, and*
2. *if $N_i \subseteq N^{out}$, $(r_i, v) \in \mathcal{K}$.*

Proof. Let $i \in [\ell]$, and suppose $N_i \subseteq N^{in}$. Since v and r_i are in the same connected component in $G(\mathcal{I}_{\mathcal{L}})$, there is some minimal sequence of commodity pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_p, t_p)$ such that $v \in \{s_1, t_1\}$, $r_i \in \{s_1, t_1\}$, and $|\{s_i, t_i\} \cap \{s_{i+1}, t_{i+1}\}| = 1$ for all $i \in [p-1]$. By definition of r_i, it follows that $t_p = r_i$. For a contradiction, suppose $p > 2$.

Consider the final two commodities, (s_{p-1}, t_{p-1}) and (s_p, t_p). Since the sequence is minimal, $s_p = s_{p-1}$ or $s_p = t_{p-1}$, and neither s_{p-1} nor t_{p-1} is equal to t_p.

Suppose $s_p = s_{p-1}$. Then t_{p-1} is on the unique dipath between s_{p-1} and t_p and $t_{p-1}t_p \in \texttt{cl}(D)$. This scenario is shown in Figure 7.49. Since $s_p \in \mathcal{S}(t_{p-1}) \cap \mathcal{S}(t_p)$, it follows that $(t_{p-1}, t_p) \in \mathcal{K}$. By minimality, either $p = 2$ and $t_{p-1} = v$ or $p > 2$ and $|\{s_{p-2}, t_{p-2}\} \cap t_{p-1}| = 1$. In either case, replacing (s_{p-1}, t_{p-1}) and (s_p, t_p) with (t_{p-1}, t_p) gives a smaller sequence of commodities connecting v and r_i in $G(\mathcal{I}_{\mathcal{L}})$, a contradiction.

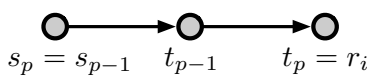


Figure 7.49: $s_p = s_{p-1}$.

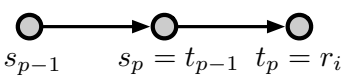


Figure 7.50: $s_p = t_{p-1}$.

Suppose instead that $s_p = t_{p-1}$ (Figure 7.50). Then since $s_{p-1}t_p \in \texttt{cl}(D)$ and $s_p \in \mathcal{T}(s_{p-1}) \cap \mathcal{S}(t_p)$, there is a commodity $(s_{p-1}, t_p) \in \mathcal{K}$. By minimality, either $p = 2$ and $s_{p-1} = v$ or $p > 2$ and $|\{s_{p-2}, t_{p-2}\} \cap s_{p-1}| = 1$. In either case, replacing (s_{p-1}, t_{p-1}) and (s_p, t_p) with (s_{p-1}, t_p) gives a smaller sequence of commodities connecting v and r_i in $G(\mathcal{I}_\mathcal{L})$, a contradiction. Therefore $p = 1$ and $(v, r_i) \in \mathcal{K}$ as desired. The case where $N_i \subseteq N^{out}$ follows by an analogous argument. □

In the following lemma, the sets $\mathcal{S}_\mathcal{J}$ and $\mathcal{T}_\mathcal{J}$ represent the set of sources and sinks for the instance $\phi_\mathcal{N}(\mathcal{I}_\mathcal{J})$. Similarly, for all $v \in N$, $\mathcal{S}_\mathcal{J}(v) = \{s : (s, v) \in \phi_{\mathcal{N},R}(\mathcal{K}_\mathcal{J})\}$ and $\mathcal{T}_\mathcal{J}(v) = \{t : (v, t) \in \phi_{\mathcal{N},R}(\mathcal{K}_\mathcal{J})\}$.

Lemma 7.39. *Let $\mathcal{I}$ be a maximal merged-tree instance. Let $\mathcal{N}$ be the node sets of the maximally-connected components in $G(\mathcal{I}_\mathcal{L})$ and let R be the set of representative nodes. Then $\phi_\mathcal{N}(\mathcal{I}_\mathcal{J})$ satisfies the following conditions.*

1. *for all distinct $s, s' \in \mathcal{S}_\mathcal{J} \setminus \{r\}$, $ss' \in \texttt{cl}(D) \Rightarrow \mathcal{T}_\mathcal{J}(s) \cap \mathcal{T}_\mathcal{J}(s') = \emptyset$;*
2. *for all distinct $t, t' \in \mathcal{T}_\mathcal{J} \setminus \{r\}$, $tt' \in \texttt{cl}(D) \Rightarrow \mathcal{S}_\mathcal{J}(t) \cap \mathcal{S}_\mathcal{J}(t') = \emptyset$.*

Proof. Let $s, s' \in \mathcal{S}_\mathcal{J} \setminus \{r\}$ be distinct nodes, and suppose $ss' \in \texttt{cl}(D)$. For a contradiction, suppose $\mathcal{T}_\mathcal{J}(s) \cap \mathcal{T}_\mathcal{J}(s') \neq \emptyset$. That is, there are commodities (s, t) and (s', t) in $\phi_\mathcal{N}(\mathcal{K}_\mathcal{J})$. By definition of $\phi_\mathcal{N}(\mathcal{K}_\mathcal{J})$, it follows that instance $\mathcal{I}$ contains commodities (s_1, t_1) and (s_2, t_2) where $\phi_{\mathcal{N},R}(s_1) = s$, $\phi_{\mathcal{N},R}(s_2) = s'$ and $\phi_{\mathcal{N},R}(t_1) = \phi_{\mathcal{N},R}(t_2) = t$. Since $\mathcal{I}$ is maximal, by Lemma 7.38, $\mathcal{K}$ contains the commodity pairs (s_1, s), (s_2, s'), (t, t_1), and (t, t_2). Thus, the graph $\texttt{cl}(D)$ must contain all of the arcs in $\texttt{cl}(D')$, where D' is given in Figure 7.51.

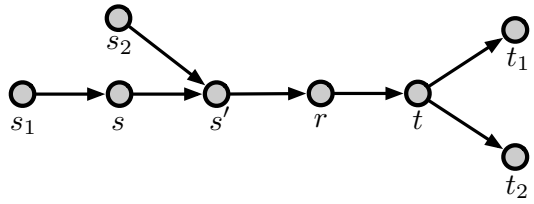


Figure 7.51

We have the following chain of commodities that must be in $\mathcal{K}$, since $(s_1, t_1), (s_2, t_2), (s_1, s)$, $(s_2, s'), (t, t_1)$, and (t, t_2) are all in $\mathcal{K}$.

$$s_1 \in \mathcal{S}(s) \cap \mathcal{S}(t_1), st_1 \in \texttt{cl}(D) \Rightarrow (s, t_1) \in \mathcal{K}$$
$$t_1 \in \mathcal{T}(s) \cap \mathcal{T}(t), st \in \texttt{cl}(D) \Rightarrow (s, t) \in \mathcal{K}$$
$$s_2 \in \mathcal{S}(s') \cap \mathcal{S}(t_2), s't_2 \in \texttt{cl}(D) \Rightarrow (s', t_2) \in \mathcal{K}$$
$$t_2 \in \mathcal{T}(s') \cap \mathcal{T}(t), s't \in \texttt{cl}(D) \Rightarrow (s', t) \in \mathcal{K}$$
$$t \in \mathcal{T}(s) \cap \mathcal{T}(s'), ss' \in \texttt{cl}(D) \Rightarrow (s, s') \in \mathcal{K}$$

However, $(s, s') \in \mathcal{K}$ contradicts that s and s' are in different connected components in $G(\mathcal{I}_{\mathcal{L}})$. Therefore the first condition holds. The second condition is analogous. □

We obtain the following corollary for spider instances.

Corollary 7.40. *Let $\mathcal{I}$ be a maximal spider instance. Let $\mathcal{N}$ be the node sets of the maximally-connected components in $G(\mathcal{I}_{\mathcal{L}})$. Then $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$ is a simple spider instance.*

It follows that in order to solve spider instances of `min-cardinality-SPP`, it remains to find an algorithm that solves simple spider instances. Recall that any subgraph H of $\texttt{cl}(D)$ when D is a merged-tree splits into $H = H|_{\mathcal{L}} \cup H|_{\mathcal{J}}$. We will prove that simple spider instances can be solved with the same approach as the algorithm for solving star instances. This result relies on first proving that there is a solution H where $H|_{\mathcal{L}}$ has no arcs, and $H = H|_{\mathcal{J}}$.

Lemma 7.41. *Let $\mathcal{I} = (D, \mathcal{K})$ be a simple spider instance. There is an optimal solution H for $\mathcal{I}$ such that $H = H|_{\mathcal{J}}$.*

Proof. Let $\mathcal{I}$ be a simple spider instance. Since $\mathcal{I}$ is also a junction tree instance, $\mathcal{K}_{\mathcal{L}} = \emptyset$, and the trivial graph $H_1 = (N, \emptyset)$ is an optimal solution for $\mathcal{I}_{\mathcal{L}}$. By Lemma 7.36, there is an optimal solution H where $H|_{\mathcal{J}} \subseteq \texttt{cl}(D)[V]$, where V is the set of representative nodes of the maximally-connected components in $H|_{\mathcal{L}}$. Suppose H is an optimal solution of this form with the fewest arcs in $H|_{\mathcal{L}}$. For a contradiction, suppose there is some non-singleton component in $H|_{\mathcal{L}}$.

Let $H' = (N', A')$ be a maximally-connected component of H that contains a non-singleton component in $H|_{\mathcal{L}}$. Let $\mathcal{K}' \subseteq \mathcal{K}$ be the set of commodities for which H' contains an s_k, t_k-dipath. Since H' is weakly-connected, it has at least $|N'| - 1$ arcs. Let H'' be the subgraph with arc set $\{sr : s \in \mathcal{S} \cap N'\} \cup \{rt : t \in \mathcal{T} \cap N'\}$. Observe that $\bar{H} = H - H' + H''$ is a feasible solution, and there are fewer arcs in $\bar{H}|_{\mathcal{L}}$ than in $H|_{\mathcal{L}}$. Additionally, $\bar{H}|_{\mathcal{J}} \subseteq \texttt{cl}(D)[\bar{V}]$, where $\bar{V}$ is the set of representative nodes of the maximally-connected components in $\bar{H}|_{\mathcal{L}}$. The

number of arcs in H'' is $|N'|$ if $r \notin N'$, and $|N'| - 1$ if $r \in N'$. Thus, the number of arcs in H' must be $|N'| - 1$ and $r \notin N'$ by choice of H. Moreover, the underlying undirected graph of H' forms a tree on N'.

Let $\mathcal{V} = \{V_1, V_2, \ldots, V_p\}$ be the node sets of the maximally-connected components in $H'|_{\mathcal{L}}$. For each $i \in [p]$, let w_i denote a node that is *furthest* from r in D, chosen arbitrarily. Note that if $v_i \neq w_i$, then w_i must have degree one in H'. An example is given in Figure 7.52, where the solid arcs are those in D, and the dashed arcs make up the component H'.

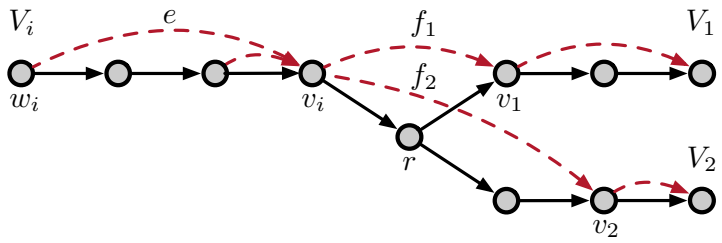


Figure 7.52

By choice of H', there is some non-singleton component in $H'|_{\mathcal{L}}$ with node set V_i, representative node v_i where $v_i \neq w_i$. Without loss of generality, suppose $V_i \subseteq N^{in}$. Then w_i is a source node, and there is a unique arc $e = w_i v$ incident to w_i in H. By optimality, $H \setminus e$ is infeasible, so some commodity with source w_i reaches its sink via an arc departing v_i. Let $\mathcal{K}_1 = \{k \in \mathcal{K}' : s_k = w_i\}$ and let $\mathcal{K}_2 = \{k \in \mathcal{K}' : s_k \in V_i \setminus w_i\}$. We define the following sets, A_1 and A_2, where

$$A_1 = \{f \in \delta_H^+(v_i) : H \setminus f \text{ has no } s_k, t_k\text{-dipath for some } k \in \mathcal{K}_1\}, \text{ and}$$
$$A_2 = \{f \in \delta_H^+(v_i) : H \setminus f \text{ has no } s_k, t_k\text{-dipath for some } k \in \mathcal{K}_2\}.$$

That is, A_1 is the set of arcs in $\delta_H^+(v_i)$ that are required to route some commodity in $\mathcal{K}_1$, and A_2 is the set of arcs in $\delta_H^+(v_i)$ that are required to route some commodity in $\mathcal{K}_2$. Let $\bar{A}_1$ be the set obtained by mapping each arc in A_1 to depart w_i instead of v_i. That is, $\bar{A}_1 = \{w_i v : uv \in A_1\}$.

Let $\bar{H}'$ be the graph obtained from H' by removing arcs $A_1 \setminus A_2$ and e, and adding the set $\bar{A}_1$. Observe that $\bar{H}'$ is feasible for $\mathcal{K}'$, $\bar{H}'|_{\mathcal{L}}$ has one fewer arc than $H'|_{\mathcal{L}}$, and maintains that all arcs in $\texttt{cl}(D) \cap \bar{H}'$ are between representative nodes of components in $\bar{H}'|_{\mathcal{L}}$. If $|A_1 \cap A_2| \leq 1$, then the graph $(H \setminus H') \cup \bar{H}'$ contradicts the choice of H as it would also be optimal with the desired structure, and have fewer arcs from $\texttt{cl}(D)|_{\mathcal{L}}$. Therefore, $|A_1 \cap A_2| \geq 2$. Thus, there are at least two arcs in $\delta_H^+(v_i)$, required to route commodities both in $\mathcal{K}_1$ and $\mathcal{K}_2$, as depicted by arcs f_1 and f_2 in Figure 7.52.

That is, there are at least two node sets V_1 and V_2, with $V_1, V_2 \subseteq N^{out}$, such that $\mathcal{T}(w_i) \cap V_j \neq \emptyset$ and $\mathcal{T}(V_i \setminus \{w_i\}) \cap V_j \neq \emptyset$ for each $j \in [2]$. Since instance $\mathcal{I}$ satisfies properties (K1) and (K2), no pair of sources in V_i shares the same sink. As a result, $|V_1|$ and $|V_2|$ each have cardinality at least 2. By an analogous argument, each of the representative nodes v_1 and v_2 must have degree 2 in the graph $H'|_{\mathcal{J}}$. Continuing in this manner forces there to be a cycle in the graph $H'|_{\mathcal{J}}$, contradicting that H' is acyclic. Therefore, $H|_{\mathcal{L}}$ must only have singleton components, and $H = H|_{\mathcal{J}}$ as required. □

Lemma 7.42. *Let $\mathcal{I} = (D, \mathcal{K})$ be a simple spider instance of* `min-cardinality-SPP`*. There is an optimal solution such that each component either has arc set $\{s_k t_k : k \in \mathcal{K}'\}$, or $\{s_k r : k \in \mathcal{K}'\} \cup \{r t_k : k \in \mathcal{K}'\}$ for some $\mathcal{K}' \subseteq \mathcal{K}$.*

Proof. For ease of notation, we will call a subgraph *type I* if it has arc set $\{s_k t_k : k \in \mathcal{K}'\}$ for some $\mathcal{K}' \subseteq \mathcal{K}$, *type II* if it has arc set $\{s_k r : k \in \mathcal{K}'\} \cup \{r t_k : k \in \mathcal{K}'\}$ for some $\mathcal{K}' \subseteq \mathcal{K}$, and *type III* otherwise.

Let H be an optimal solution such that $H = H|_{\mathcal{J}}$, which exists due to Lemma 7.41. Moreover, let H be an optimal solution of this form with the fewest maximally-connected components of type III. For a contradiction, suppose there is some component, $H' = (N', A')$, that is of type III. Let $\mathcal{K}' \subseteq \mathcal{K}$ be the set of commodities served by H'. By the same logic as in the proof of Lemma 7.27, H' must have $|N'| - 1$ arcs, and cannot contain r. Otherwise, replacing H' in H with a subgraph of type II would give an optimal solution with fewer type III components. Let uv be an arc in H'. Since there are no arcs with both endpoints on the same leg, H' cannot contain any arc uv where $uv \neq s_k t_k$ for all $k \in \mathcal{K}$, as otherwise it could be removed while maintaining feasibility, contradicting the optimality of H. Thus, H is in fact a type I graph, contradicting that it was type III. □

The proof of the following corollary follows analogously to the proof that an optimal solution to a star instance is equal to the union of optimal solutions to each weakly-connected subinstance. We repeat the proof for completeness.

Corollary 7.43. *Let $\mathcal{I} = (D, \mathcal{K})$ be a simple spider instance of* `min-cardinality-SPP`*. Then $m^* = \sum_{i \in [\ell]} m_i^*$.*

Proof. Let $H = (N, A')$ be an optimal solution for instance $\mathcal{I}$, such that each component either has arc set $\{s_k t_k : k \in \mathcal{K}'\}$, or $\{s_k r : k \in \mathcal{K}'\} \cup \{r t_k : k \in \mathcal{K}'\}$ for some $\mathcal{K}' \subseteq \mathcal{K}$. Let $\{N_1, N_2, \ldots, N_\ell\}$ denote the node sets of the maximally-connected components in $G(\mathcal{I})$, and let $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell\}$ be the partition of $\mathcal{I}$ into weakly-connected subinstances.

Let $i \in [\ell]$, and let H_i be the subgraph of H which includes all arcs with an endpoint in $N_i \setminus \{r\}$. That is, $H_i = (N, A_i)$ where $A_i := \{uv \in A' : \{u, v\} \cap (N_i \setminus \{r\}) \neq \emptyset\}$. For each commodity $k \in \mathcal{K}_i$, the s_k, t_k-dipath in H consisted of at most two arcs, each with at least one endpoint in $\{s_k, t_k\}$ which was not equal to r. Therefore H_i is feasible for instance $\mathcal{I}_i$.

Moreover, for each $j \neq i$, if an arc uv is in A_i and A_j, then wlog u is a source node in $\mathcal{I}_i$ and v is sink node in $\mathcal{I}_j$, each not equal to r. However, such an arc could be removed while maintaining feasibility. Thus since H is an optimal solution, no such arc exists and $A_i \cap A_j = \emptyset$. As a result, $m^* \geq \sum_{i \in [\ell]} m_i^*$, and by Lemma 7.6 we have that $m^* = \sum_{i \in [\ell]} m_i^*$. □

We now state the algorithm for solving simple spider instances. Observe that the algorithm is in fact the same as the algorithm for solving star instances.

Algorithm 22: `Simple-spider`$(\mathcal{I})$

Input: $\mathcal{I} = (D, \mathcal{K})$, a simple spider instance of `min-cardinality-SPP`.

$\{N_1, N_2, \ldots, N_\ell\} \leftarrow$ node sets of the maximally-connected components in $G(\mathcal{I})$
$\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell\}$ be the corresponding partition of $\mathcal{I}$
$A' = \emptyset$
for $i \in [p]$ **do**
    if $|V_i \cup r| - 1 \leq |\mathcal{K}_i|$ **then**
        $A' \leftarrow A' \cup \{s_k r : k \in \mathcal{K}_i\} \cup \{r t_k : k \in \mathcal{K}_i\}$
    else
        $A' \leftarrow A' \cup \{s_k t_k : k \in \mathcal{K}_i\}$
return $H = (N, A')$

Combining Lemma 7.42 and Corollary 7.43 gives the following result.

Proposition 7.44. *Algorithm 22 returns an optimal solution given a simple spider instance in polynomial time.*

Therefore, the following algorithm returns an optimal solution given a spider instance in polynomial time.

Algorithm 23: `Spider`$(\mathcal{I})$

Input: $\mathcal{I} = (D, \mathcal{K})$, a spider instance of `min-cardinality-SPP`.

$\mathcal{I} = (D, \mathcal{K}) \leftarrow$ `Augment`$(\mathcal{I})$
$\mathcal{I}_{\mathcal{L}} \leftarrow (D, \mathcal{K}_{\mathcal{L}})$
$\mathcal{I}_{\mathcal{J}} \leftarrow (D, \mathcal{K}_{\mathcal{J}})$
$\mathcal{N} \leftarrow \{N_1, N_2, \ldots, N_\ell\}$, node sets of maximally-connected components in $G(\mathcal{I}_{\mathcal{L}})$
$H_1 \leftarrow \bigcup_{L \in \mathcal{L}}$ `in-out-tree`$(\mathcal{I}_{\mathcal{L}}[L])$
$H_2 \leftarrow$ `simple-spider`$(\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}}))$
return $H_1 \cup H_2$

Proposition 7.45. *Algorithm 23 returns an optimal solution given a spider instance in polynomial time.*

Proof. Since instance $\mathcal{I}$ and $\mathcal{I}^{max} =$ `AUGMENT`$(\mathcal{I})$ are equivalent under uncrossing, and any feasible solution for $\mathcal{I}^{max}$ is feasible for $\mathcal{I}$, it is sufficient to solve $\mathcal{I}^{max}$. Thus, we may assume $\mathcal{I}$ is maximal and otherwise we replace it with $\mathcal{I}^{max}$. By Lemma 7.36, an optimal solution to $\mathcal{I}$ is obtained by taking the union of a minimally-crossing optimal solution to $\mathcal{I}_{\mathcal{L}}$ and an optimal solution to $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. Note that H_1 is an optimal solution for $\mathcal{I}_{\mathcal{L}}$ since $\mathcal{I}_{\mathcal{L}}$ is the disjoint union of a set of in-tree and out-tree instances, each defined on a leg of D. Moreover, by construction of Algorithm 20, H_1 is minimally-crossing.

Since $\mathcal{I}$ is maximal, $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$ is a simple spider instance by Lemma 7.40. Additionally, by Proposition 7.44, H_2 is an optimal solution for $\phi_{\mathcal{N}}(\mathcal{I}_{\mathcal{J}})$. Therefore the algorithm returns an optimal solution for instance $\mathcal{I}$. □

Thus, we have proven the following theorem.

Theorem 7.46. *Spider instances of* `min-cardinality-SPP` *can be solved in polynomial time.*

Additionally, we obtain the following proposition.

Proposition 7.47. *Let $\mathcal{I} = (D, \mathcal{K})$ be a spider instance. Then $m^* = \sum_{i \in [\ell]} m_i^*$.*

Proof. Let $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell\}$ be the partition of $\mathcal{I}$ into weakly-connected subinstances. Note that the algorithm `AUGMENT` only adds a commodity (v, v') for pairs v, v' that are in the same connected component in $G(\mathcal{I})$. As a result, $G(\mathcal{I}) = G(\mathcal{I}^{max})$, and the node sets $\{N_1, N_2, \ldots, N_\ell\}$ of the maximally-connected components are the same. Let H be the

optimal solution for $\mathcal{I}$ returned by Algorithm 23. By construction, for any arc uv in H, either $u, v \in N_i$ for some $i \in [\ell]$, or one of u and v are equal to r. It follows that $H_i = (N, A_i)$ where $A_i := \{uv \in A' : \{u, v\} \cap (N_i \setminus \{r\}) \neq \emptyset\}$ is feasible for instance $\mathcal{I}_i$, and H_i and H_j are disjoint whenever $i \neq j$. Thus, $m^* = \sum_{i \in [\ell]} m_i^*$ as required. □

It is not true that junction tree instances in general can be solved with the same algorithmic approach. Consider the following junction tree instance that satisfies properties (K1) and (K2). Algorithm 22 will not return an optimal solution for this instance. In particular, while the instance is weakly-connected, D is not an optimal solution, nor is the arc set $\{s_k t_k : k \in \mathcal{K}\}$. Instead, an optimal solution is presented in Figure 7.54.

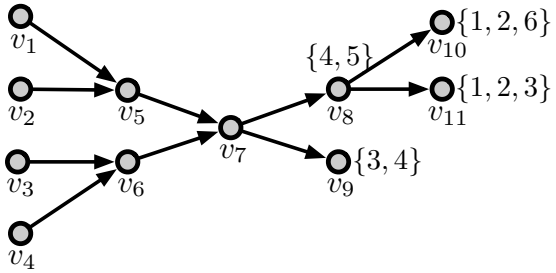


Figure 7.53

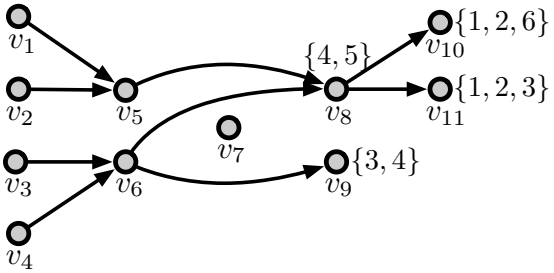


Figure 7.54

7.6 Summary and observations

In this chapter, we presented a simple 3-approximation algorithm for tree instances of `min-cardinality-SPP`, as well as combinatorial algorithms for spider instances, and in- and out-tree instances. We showed that solving merged-tree instances reduces to solving the special case of junction tree instances. Similar to the min-degree objective, finding good (potentially exact) algorithms for junction tree instances remains an open problem. Overall, a natural open problem is to determine the hardness of `min-cardinality-SPP` on general graphs, and for tree instances.

Part III

Conclusion

Chapter 8

Conclusion and Future Work

In this thesis we consider algorithms for addressing the temporal network design and sortation problems arising in networks with consolidation.

8.1 DDD results

The first focus of this thesis is the advancement of the DDD paradigm. In particular, we focus on improving DDD in its application to temporal problems in large, region-based networks. In these networks, hub nodes often have trucks departing along different outbound arcs at many times each day. In addition, consolidation often relies on utilizing limited warehouse storage capacity effectively. While these applications are our primary focus, the developments presented in this thesis have scope beyond problems defined on these specific network structures.

We demonstrate that the generic DDD approach is ill-suited to solving problems where near-optimal solutions require a large number of departure times at many high out-degree nodes in the network. We present a new arc-based DDD approach that permits an arc-dependent set of departure times to be maintained in each iteration, rather than a fixed set of departure times for each node. As a result, the lower bound formulation in each iteration is often smaller than the formulation solved in the original node-based DDD approach. Moreover, the arc-based approach builds upon the original DDD paradigm, and as a result, the modifications required to build the arc-based DDD algorithm from a node-based DDD algorithm are minimal.

The arc-based DDD approach works particularly well when the problem permits a fine arc

partitioning. However, not all problems permit a fine partitioning, even if arc departure times are largely disjoint. Consider two arcs in $\delta^+(v)$ that have independent departure times in an iteration of DDD. If the arcs must be in the same part in any valid arc partition, we saw that different departure times for the arcs would result in additional timed copies of arcs in $\delta^-(v)$. When $\delta^-(v) = \emptyset$, there is no downside to allowing disjoint departure times for arcs departing v, and the definition of a valid arc partition can be relaxed for these nodes. It is possible that when the in-degree at a node v is smaller than the out-degree, allowing arc-dependent departure times is advantageous even when it results in commodities with variables for multiple copies of the same incoming arc. Characterizing when these design choices should be made is an interesting research direction.

Open problem 8.1. *Is there a generalization of the arc-based DDD approach that permits independent departure times for arcs in the same part of a valid arc partition? Can such a policy out-perform the current arc-based DDD approach?*

In this thesis we demonstrate the necessity to relax storage constraints in the lower bound model when DDD is applied to problems with static node storage. Moreover, we highlight the need to relax storage without leading to an overly weak relaxation in each iteration, since this can cause DDD to run for many iterations before terminating. We present bounds on the additional storage that must be permitted in each iteration, relying solely on the structure of of the standard map, μ. As a result, the bounds presented generalize to fixed-charge problems, including the service network design problem.

We also present conditions on the timed node set in a partial network when DDD is applied to a problem with cyclic constraints. These conditions lead to denser partial networks. We show that reducing the size of partial networks as they grow over each iteration of DDD remains challenging. In particular, certain policies for removing timed nodes from the current partial network can result in iteration cycling (iterations repeat). This leaves the following open question.

Open problem 8.2. *Is there a timed node removal policy that does not lead to iteration cycling, and that improves the performance of DDD in practice?*

Another strategy to reduce the number of large DDD iterations is to initialize the algorithm with a partial network that results in a tighter lower bound formulation. The standard approach in DDD is to initialize the timed node set N_S to be the minimal set satisfying the given node set properties. However, this often results in many iterations that have weak relaxations. Ideally, the initial set N_S would be small, while also resulting in a strong relaxation. For instances in which there is a sufficient data, a promising approach would be to use machine learning to predict the set N_S in the final iteration of DDD.

Open problem 8.3. *Can we use machine learning to predict small timed node sets that result in strong relaxations? Does initializing DDD with such a set result in a faster algorithm than initializing DDD with the standard minimal set?*

Our work on arc-based DDD as well as incorporating cyclic constraints all surrounds the improvement of DDD to model problems where the size of the support of an optimal solution is no longer a tiny fraction of the total timed arc set. Not only do many networks behave cyclically, but in fact the flow on each arc is roughly constant for periods of the day, i.e., they are temporally-repeating during periods of the day. As a result, solutions can only be expressed on large subgraphs of the full time-expanded network. An interesting direction of research would be to incorporate repetitive flows into the DDD framework, without leading to large partial networks.

Open problem 8.4. *Is there a DDD-type approach that can be applied to problems with temporally-repeating flows?*

It is possible that in order for a DDD approach to perform well on problems with large supports, the upper bound procedure needs to be strengthened. One direction to improve upper bounds is to study the strength of continuous formulations compared to discrete formulations for various problems. The continuous upper bound formulation presented in Chapter 2 no longer applies to packet routing, nor to SND with cyclic constraints. An interesting question would be to develop a continuous formulation for these problems, and compare its performance to standard discrete models.

Open problem 8.5. *Are there upper bound and refinement procedures that when combined with the lower bound model in Section 5.1.2 outperform the approach of solving the full time-indexed formulation directly?*

Finally, when faced with a temporal network design problem, ideally we would be able to predict whether or not a DDD algorithm would outperform the approach of directly solving the full time-indexed formulation, without having to implement and test both strategies. One supporting measurement for this goal would be to bound the number of iterations DDD could require until terminating. This suggests the following open problem.

Open problem 8.6. *Can we develop bounds on the number of iterations required until DDD terminates, based on the structure of the flat network D?*

8.2 Sortation results

The second focus of this thesis is the modelling and optimization of sortation requirements. We present an abstract model defined on the transitive closure $\texttt{cl}(D)$ of the physical network D. We show that an assignment of sort points is expressed by a subgraph H of $\texttt{cl}(D)$, and the out-degree of a node in H is the number of sort points assigned to the corresponding warehouse. We prove that it is NP-hard to determine whether a feasible sortation plan exists given fixed sort point capacity limits at each warehouse. We consider the natural problems of minimizing the max out-degree of a feasible solution, and minimizing the number of arcs in a feasible solution. Our work focuses on tree instances for each objective.

In Chapter 6, we consider the problem of minimizing the maximum number of sort points required at a warehouse. We first present combinatorial lower bounds similar to those used for degree-bounded spanning trees and arborescences in earlier work. We develop a fast algorithm for solving single-source instances, and an additive 1-approximation algorithm for multi-source out-tree instances. A natural open question remains.

Open problem 8.7. *Can out-tree instances of* `min-degree-SPP` *be solved in polynomial time?*

For the multi-source setting, the lower bound formulation has an inherent gap of at least one for out-tree instances. For the star setting, for any $\epsilon > 0$, there is an instance for which the largest lower bound is only $\frac{2}{3}\Delta^* + \epsilon$. In this thesis, we present a 2-approximation for star instances and are left with the following question.

Open problem 8.8. *Is there a polytime α-approximation algorithm for star instances of* `min-degree-SPP` *for some $\alpha < 2$?*

We also present an approach for obtaining an approximation algorithm for tree instances given an approximation algorithm for junction tree instances.

Open problem 8.9. *Is there an $O(1)$-approximation algorithm for junction tree instances of* `min-degree-SPP`*? An $O(1)$-approximation algorithm for tree instances?*

In Chapter 7, we present algorithms for minimizing the total number of sort points required in a network. One of the main questions remaining for this objective is the following.

Open problem 8.10. *Is* `min-cardinality-SPP` *polytime solvable?*

In this thesis we present combinatorial algorithms for in-tree, out-tree, and spider instances.

We showed that in order to find an algorithm for merged-tree instances, it remains to find an algorithm for junction tree instances.

Open problem 8.11. *Is there a polytime algorithm for junction tree instances of* `min-cardinality-SPP`*?*

Each of the algorithms presented is combinatorial, and for each setting considered, the union of optimal solutions for each weakly-connected subinstance is an optimal solution. An interesting direction would be to consider instance classes for which this decomposition no longer gives an optimal solution.

In practical applications, while often sparse, the network structure is likely to be more complex than a tree. Furthermore, while the objectives considered in this thesis are natural initial choices for understanding the theory of sort point application, they are not exhaustive. The procedures of sorting and cross-docking have different costs, and require different amounts of time. Future work would be to find approximate and exact solution approaches for sort point problems incorporating speed and cost, on general digraphs.

References

[1] N. Bansal, R. Khandekar, and V. Nagarajan. Additive guarantees for degree-bounded directed network design. *SIAM J. Comput.*, 39(4):1413–1431, 2009.

[2] N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017.

[3] N. Boland and M. Savelsbergh. Perspectives on integer programming for time-dependent models. *TOP - invited paper*, 27:147–173, 2019.

[4] J. T. Bowen. A spatial analysis of fedex and ups: hubs, spokes, and network structure. *Journal of Transport Geography*, 24:419–431, 2012.

[5] C. Busch, M. Magdon-Ismail, M. Mavronicolas, and P. G. Spirakis. Direct routing: Algorithms and complexity. In *ESA*, 2004.

[6] P. M. Castro, A. P. Barbosa-Póvoa, and H. A. Matos. Optimal periodic scheduling of batch plants using rtn-based discrete and continuous-time formulations: A case study approach. *Industrial & Engineering Chemistry Research*, 42(14):3346–3360, 2003.

[7] K. Chaudhuri, S. Rao, S. J. Riesenfeld, and K. Talwar. A push-relabel algorithm for approximating degree bounded MSTs. In *ICALP (1)*, volume 4051 of *Lecture Notes in Computer Science*, pages 191–201. Springer, 2006.

[8] K. Chaudhuri, S. Rao, S. J. Riesenfeld, and K. Talwar. What would Edmonds do? Augmenting paths and witnesses for degree-bounded MSTs. *Algorithmica*, 55(1):157–189, 2009.

[9] V. Chvátal. Tough graphs and hamiltonian circuits. *Discrete Math.*, 5:215–228, 1973.

[10] T. G. Crainic. Service network design in freight transportation. *European Journal of Operations Research*, 122(2):272–288, 2000.

[11] T. G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1):73–99, 2001.

[12] T.G. Crainic and M. Hewitt. *Network Design with Applications to Transportation and Logistics*, chapter 12, pages 347–382. Editors T.G. Crainic, M. Gendreau, and B. Gendron, Springer, 2021.

[13] S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.

[14] S. Dehghani, S. Ehsani, M. Hajiaghayi, and V. Liaghat. Online degree-bounded steiner network design. In *SODA*, pages 164–175. SIAM, 2016.

[15] M. Di Ianni. Efficient delay routing. *Theoretical Computer Science*, 196:131–151, 1998.

[16] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 1998.

[17] L. Fleischer and M. Skutella. Minimum cost flows over time without intermediate storage. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 66–75, 2003.

[18] L. Fleischer and M. Skutella. Quickest flows over time. *SIAM Journal on Computing*, 36(6):1600–1630, 2007.

[19] L.R. Ford and D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, pages 419–433, 1958.

[20] L.R. Ford and D.R. Fulkerson. Flows in networks. *Princeton University Press*, 1962.

[21] M. Fürer and B. Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *Journal of Algorithms*, 17:409–423, 1994.

[22] H. Gabow. Maximum cardinality f-matching in time $o(n^{2/3}m)$, https://arxiv.org/pdf/2311.14236, 2023.

[23] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[24] M. X. Goemans. Minimum bounded degree spanning trees. In *FOCS*, pages 273–282. IEEE Computer Society, 2006.

[25] X. Guo, G. Kortsarz, B. Laekhanukit, S. Li, D. Vaz, and J. Xian. On approximating degree-bounded network design problems. *Algorithmica*, 84(5):1252–1278, 2022.

[26] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[27] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2719:397–409, 2003.

[28] E. He, N. Boland, G. Nemhauser, and M. Savelsbergh. A dynamic discretization discovery algorithm for the minimum duration time-dependent shortest path problem. *International conference on the integration of constraint programming, artificial intelligence, and operations research (CPAIOR)*, pages 289–297, 2018.

[29] E. He, N. Boland, G. Nemhauser, and M. Savelsbergh. An exact algorithm for the service network design problem with hub capacity constraints. *Networks*, 80(4):572–596, 2022.

[30] M. Hewitt. Enhanced dynamic discretization discover for the continuous time load design problem. *Transportation Science*, 53:1731–1750, 2019.

[31] M. Hewitt. The flexible scheduled service network design problem. *Transportation Science*, 56:1000–1021, 2022.

[32] A. Jassy. 2022 letter to shareholders, https://www.aboutamazon.com/news/company-news/amazon-ceo-andy-jassy-2022-letter-to-shareholders, 2023.

[33] J. Kleinberg. The small-world phenomenon: an algorithmic perspective. *Symposium on Theory of Computing (STOC)*, pages 163–170, 2000.

[34] J. Koenemann and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM Journal on Computing*, 31(6):1783–1793, 2002.

[35] J. Koenemann and R. Ravi. Quasi-polynomial time approximation algorithm for low-degree minimum-cost steiner trees. In *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 289–301. Springer, 2003.

[36] J. Koenemann and R. Ravi. Primal-dual meets local search: Approximating MSTs with nonuniform degree bounds. *SIAM J. Comput.*, 34:763–773, 01 2005.

[37] G. Kortsarz and Z. Nutov. Bounded degree group steiner tree problems. In *IWOCA*, volume 12126 of *Lecture Notes in Computer Science*, pages 343–354. Springer, 2020.

[38] D. Kozen. *The design and analysis of algorithms*. Springer Science & Business Media, 1992.

[39] F. Lagos, N. Boland, and M. Savelsbergh. Dynamic discretization discovery for solving the continuous time inventory routing problem with out-and-back routes. *Computers & Operations Research*, 141, 2022.

[40] C. L. Lara, J. Koenemann, Y. Nie, and C. C. de Souza. Scalable timing-aware network design via lagrangian decomposition. *European Journal of Operational Research*, 309(1):152–169, 2023.

[41] L.C. Lau, J. Naor, M. R. Salavatipour, and M. Singh. Survivable network design with degree or order constraints. *SIAM J. Comput.*, 39(3):1062–1087, 2009.

[42] L.C. Lau and M. Singh. Additive approximation for bounded degree survivable network design. *SIAM J. Comput.*, 42(6):2217–2242, 2013.

[43] F.T. Leighton, B.M. Maggs, and S.B Rao. Packet routing and job shop scheduling in O(congestion+dilation) steps. *Combinatorica*, 14:167–186, 1994.

[44] F.T. Leighton, B.M. Maggs, and A. Richa. Fast algorithms for finding O(congestion+dilation) packet routing schedules. *Combinatorica*, 196:375–401, 1999.

[45] M. Marshall, M. Hewitt, N. Boland, and M. Savelsbergh. Interval-based dynamic discretization discovery for solving the continuous-time service network design problem. *Transportation Science*, 55 (1), 2020.

[46] S. Milgram. The small world problem. *Psychology Today*, 1(61), 1967.

[47] Z. Nutov. Approximating directed weighted-degree constrained networks. *Theoretical Computer Science*, 412(8):901–912, 2011.

[48] B. Peis, M. Skutella, and A. Wiese. Packet routing: complexity and algorithms. *Workshop on approximation and online algorithms, LNCS*, pages 217–228, 2009.

[49] B. Peis and A. Wiese. Universal packet routing with arbitrary bandwidths and transit times. *Günlük O., Woeginger G.J. (eds) Integer Programming and Combinatorial Optimization (IPCO) 2011*, 6655:362–375, 2011.

[50] S. Pottel and A. Goel. Scheduling activities with time-dependent durations and resource consumptions. *European Journal of Operations Research*, 2021.

[51] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt. Many birds with one stone: multi-objective approximation algorithms. In *STOC*, pages 438–447. ACM, 1993.

[52] R. Ravi and M. Singh. Delegate and conquer: An LP-based approximation algorithm for minimum degree MSTs. In *ICALP (1)*, volume 4051 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2006.

[53] C. Scheideler. Universal routing strategies for interconnection networks. *LNCS*, 1390:57–71, 1998.

[54] Y. O. Scherr, M. Hewitt, B. A. Neumann Saavedra, and D. C. Mattfeld. Dynamic discretization discovery for the service network design problem with mixed autonomous fleets. *Transportation Research Part B*, 141:164–195, 2020.

[55] S. Shu, Z. Xu, and R. Baldacci. Incorporating holding costs in continuous-time service network design: New model, relaxation, and exact algorithm. *Transportation Science*, 58(2):412–433, 2024.

[56] M. Singh and L.C. Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *STOC*, STOC '07, page 661–670, New York, NY, USA, 2007. Association for Computing Machinery.

[57] M. Skutella. An introduction to network flows over time. *Research Trends in Combinatorial Optimization: Bonn 2008*, pages 451–482, 2009.

[58] J. Van Belle, P. Valckenaers, and D. Cattrysse. Cross-docking: State of the art. *Omega*, 40(6):827–846, 2012. Special Issue on Forecasting in Management Science.

[59] M. Van Dyk, K. Klause, J. Koenemann, and N. Megow. Fast combinatorial algorithms for efficient sortation, https://arxiv.org/abs/2311.05094. In *IPCO*, 2024.

[60] M. Van Dyk and J. Koenemann. Dynamic discretization discovery under hard node storage constraints, https://arxiv.org/abs/2303.01419, 2023.

[61] M. Van Dyk and J. Koenemann. Sparse dynamic discretization discovery via arc-dependent time discretizations, https://arxiv.org/abs/2305.19176, 2023.

[62] D. M. Vu, M. Hewitt, N. Boland, and M. Savelsbergh. Dynamic discretization discovery for solving the time dependent traveling salesman problem with time windows. *Transportation Science*, 54(3):703–720, 2019.

[63] X. Wang and A. Regan. Local truckload pickup and delivery with hard time window constraints. *Transportation Research Part B: Methodological*, 36(2):97–112, 2002.

[64] X. Wang and A. Regan. On the convergence of a new time window discretization method for the traveling salesman problem with time window constraints. *Computers and Industrial Engineering*, 56(1):161–164, 2009.

[65] S. Win. On a connection between the existence of k-trees and the toughness of a graph. *Graphs and Combinatorics*, 5:201–205, 1989.

[66] G. Yao, D. Zhu, H. Li, and S. Ma. A polynomial algorithm to compute the minimum degree spanning trees of directed acyclic graphs with applications to the broadcast problem. *Discret. Math.*, 308(17):3951–3959, 2008.

[67] K. Zheng. Sort optimization and allocation planning in Amazon middle mile network. Informs, 2021.