# EFFICIENT DYNAMIC NETWORK FLOW ALGORITHMS

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Bruce Edward Hoppe

August 1995

EFFICIENT DYNAMIC NETWORK FLOW ALGORITHMS

Bruce Edward Hoppe, Ph.D.

Cornell University 1995

Dynamic network flows model transportation. A dynamic network consists of a graph with capacities and transit times on its edges. Flow moves through a dynamic network over time. Edge capacities restrict the rate of flow and edge transit times determine how long each unit of flow spends traversing the network. Dynamic network flows have been studied extensively for decades.

This thesis introduces the first polynomial algorithms to solve several important dynamic network flow problems. We solve them by computing chain-decomposable flows, a new class of structured dynamic flows.

We solve the quickest transshipment problem. An instance of this problem consists of a dynamic network with several sources and sinks. Each source has a specified supply and each sink a specified demand of flow. The goal is to move the appropriate amount of flow out of each source and into each sink within the least overall time. Previously, this problem could only be solved efficiently in the special case of a single source and single sink.

Our quickest transshipment algorithm depends on efficient solutions to the dynamic transshipment problem and the lexicographically maximum dynamic flow problem. The former is a version of the quickest transshipment problem in which the time bound is specified. The latter is a maximum flow problem in a dynamic

network with prioritized sources and sinks; the goal is to maximize the amount of flow leaving each high-priority subset of sources and sinks.

We also consider the universally maximum dynamic flow problem. A universally maximum dynamic flow sends flow between a source and sink so that the sink receives flow as quickly as possible; subject to that, the source releases flow as late as possible. We describe the first polynomial algorithm to approximate a universally maximum dynamic flow within a factor of $(1+\epsilon)$, for any $\epsilon > 0$. We also describe the first polynomial algorithm to compute the value of a universally maximum dynamic flow at a single specified moment of time.

# Biographical Sketch

Bruce Edward Hoppe was born to William and Patricia Hoppe on February 16, 1967, in Bethlehem, Pennsylvania. Following three years behind the footsteps of his sister Margat, Bruce attended Hanover Elementary School, East Hills Junior High School, and Freedom Senior High School, where he graduated in 1985.

Bruce attended Princeton University, where he split his time between his studies and his bicycle. Bruce raced for the Princeton Cycling Team for four years and captained the team in 1987-88. He obtained a B.S.E. in Computer Science in 1989.

After graduating from Princeton, Bruce worked two years for Princeton Transportation Consulting Group, Inc. He led the development of SUPERSPIN, a strategic optimization program for less-than-truckload trucking companies.

Bruce left PTCG in 1991 to pursue a Ph.D. in Computer Science at Cornell University. Bruce resumed racing bikes for his first two years at Cornell but then retired from collegiate cycling to focus on studying. He obtained an M.S. in Computer Science in January 1995. Upon completion of his Ph.D., Bruce plans to return to PTCG.

To Elizabeth, Christine, and Stephanie

# Acknowledgements

I am especially grateful to my advisor, Éva Tardos. We collaborated closely on all aspects of this thesis. Éva has always been generous in sharing her insights, knowledge, experience, and time with me. I have enjoyed her untiring support through my academic and personal journey at Cornell. I hope not to forget the lessons I have learned here.

Many thanks also go to the other members of my special committee, Jim Renegar and Steve Vavasis, for their inspiring courses, helpful advice, letters of recommendation, and proofreading efforts.

My first three years at Cornell were funded by a National Science Foundation Graduate Research Fellowship. I very much appreciated this financial support and the freedom it afforded me to pursue my own interests. My research was also partially funded by a Cornell University Sage Fellowship and by the National Science Foundation Presidential Young Investigator Award of Éva Tardos.

Despite my previous background in transportation modeling, I did not study dynamic network flows until my second year at Cornell, when I read a paper coauthored by Bettina Klinz. Our subsequent e-mail collaboration helped get my research off the ground, and a conversation we had in her office months later inspired the universally maximum dynamic flow snapshot algorithm of Chapter 5. In the interim, Monika Rauch and I enjoyed a series of technical discussions that contributed to the first polynomial integral evacuation algorithm, which eventually evolved into the quickest transshipment algorithm of Chapters 6–8.

Attempting to explain my research to colleagues has occasionally revealed gaps in my reasoning. For interesting conversations and stubborn questions that clarified my own thoughts, I am grateful to David Shmoys and Neal Young.

The Cornell University Department of Computer Science treated me very well during my years in Ithaca. In addition, two other institutions graciously hosted me in 1993. I followed Éva to the Department of Computer Science at my alma mater, Princeton University, for the spring semester of that year. Our research really got moving in New Jersey; during that time, I also appreciated the teaching and support of my former undergraduate advisor, Bob Tarjan. That summer, I followed Éva farther afield to the Department of Computer Science at her alma mater, Eötvös University, in Budapest, Hungary.

My years as a graduate student have not been easy; I would not have made it without support from God, friends, and family. For everything, I am grateful to God. For believing in me, I am grateful to Elizabeth Palmberg, Christine Cunningham, and Stephanie Clark. For being a great officemate, research collaborator, lasagna complimenter, and true friend, I am grateful to Lorenzo Alvisi. For their lessons in faith and friendship, I am grateful to Marty Carlisle, Genevieve Gacula, Rob Ghrist, Guerney Hunt, and Eric Savage. For their companionship in home and office, I am grateful to Rick Aaron, Cori Allen, Ray Fouche, Navindra Gambhir, Katherine Guo, Ron Minsky, Rod Moten, Mike Richman, and Divakar Viswanath. And finally, thanks to my family, Mom, Dad, and Margat — I love you.

# Table of Contents

# List of Figures

# Preface

It was a dark and stormy night. Jack lay motionless in his cabin — motionless except for his stomach, which was gyrating in his gut like a mad carnival ride. Jack wanted to stop the ride and get off. Maybe that lamb curry had been too rare.

"Mmm." Rachel snuggled comfortably against Jack; she was half asleep. Jack envied her cast-iron stomach. He also knew it was the only reason she could stand his cooking.

The full moon shone through the porthole. Jack could not enjoy the view, though. A real landlubber, Jack. How did he ever let Rachel talk him into this cruise? Rachel loved adventure, nature, the outdoors; she read about the Arctic Whale Watch in a travel magazine. Jack must have downed one too many daiquiris the night she asked him about it. Sure, why not spend a week cruising the waves of the North Atlantic?

No one was watching any whales now. Not even Captain Dirk Crandall, who leaned hard against his wheel to brace himself in the tumultuous sea. Even on duty, Dirk knocked back Jack Daniels like a drowning man gulps air. He said it calmed his nerves. Calm nerves might serve in a calm sea, but tonight Dirk's bleary gaze could not comprehend the frothy waves crashing against the hull. His vessel, the stately *Combitanic*, groaned under his fogged command.

The surging sea roused Rachel. She kissed Jack gently and drew imaginary patterns on his rugged torso. "Oh, man, I'm gonna lose it," Jack muttered as he bolted from their bunk to the bathroom.

Rachel had her own second thoughts about taking Jack on the whale watch. Jack's nerdy charm complemented his lean, athletic physique, but this trip was bringing out the milquetoast in him. All he wanted to do was lie in bed with a clear path to the toilet. He was driving Rachel totally stir crazy.

Rachel slipped into her white silk dressing gown and slipped out of the cabin. The ship crashed and groaned against the sea; almost all of her passengers were quietly hiding in their rooms. Rachel climbed her way toward the observation deck but made a wrong turn. Suddenly she found herself on the bridge. The full moon framed a silhouette manning the helm. Rachel was shocked to recognize the pair of broad shoulders beneath a frayed peacoat; massive hands emerged from the woolen sleeves and clamped firmly on the wheel.

"Dirk," Rachel said softly. Her breathy voice betrayed a rare moment of vulnerability. The captain spun around and moonbeams lit all he needed to see: flowing raven tresses tumbling over a shapely and sinewy female form.

"Rachel," he replied, "you've come a long way to find me here." The haze of alcohol hanging over Dirk did not soften his chiseled features, nor could it entirely mask the impression of power he projected.

"No, I..." Rachel found herself speechless in Dirk's presence. Could she ever forgive him? Years ago, Rachel thought not and hoped that time would quench the fire that Dirk had lit in her soul. She had to get over him. Dirk thought that distance and whiskey would erase Rachel, and the awful consequences of his violent temper, from his memory.

The *Combitanic* rocked in the storm. Rachel's mind spun as she lost her balance and fell into Dirk's brawny embrace. She could not resist as he pulled her close to him. His kiss filled her with a wondrous white light. "Oh, Dirk," she whispered.

Suddenly, a shot rang out. And another and another. Rivets popped in the bowels of the ship, as she crashed against an iceberg. Cold seawater poured through a rip in the hull opened by the hard ice. Two levels above the water, Jack under-stood what was happening; he retrieved his papers and left his cabin.

Dirk lay half-conscious at the helm. The impact had taken him off his feet and his head landed hard on the deck. “Get up!” Rachel sobbed over him. She could not budge him. Emergency lights and bells awakened the passengers in the decks below, but Rachel hardly noticed that the ship was sinking. She knelt beside Dirk. Dirk saw her dimly; his vision was receding into a dark tunnel. Rachel reached out for him. He felt her arms around him and struggled to hang on to her. “I love you. I’m sorry,” he said. “I forgive you,” she said. Time stood still.

Seemingly moments later, Jack entered the bridge. He hauled both of them to their feet. “There’s no time, let’s go!” he commanded. “The ship is about to go under. If we take the corridor to the left and go down the stairs, we should just make it to the last three seats on the only lifeboat left.”

“Honey, how did you...”

“I don’t have time to explain. Now go!”

Jack shoved them out of the bridge and followed them to the final lifeboat. He handed Rachel a life jacket and helped her aboard the dingy. He reached overhead to get another life jacket for Dirk.

Behind him, Dirk growled with horrible rage, “She’s MINE.” Dirk lunged at Jack with both hands ready to rend him limb from limb. He pinned Jack face-down again the railing, strangling him with an unbreakable grip.

“STOP!” Rachel ordered. Her quivering hands aimed a loaded flare gun between Dirk’s eyes. “Please, Dirk. Don’t make me do this. Control yourself.”

“Who is this wimp?” Dirk bellowed. “Nobody! You deserve a real man.” With one hand still clamped around Jack’s neck, Dirk reached underneath his coat. Rachel saw the glint of a blade in his hand and she pulled the trigger.

Dirk’s head lit up the night sky, and his piercing screams raged like the wind. He staggered back and collapsed to the deck. Rachel pulled Jack aboard the lifeboat and released its ties as flames licked the last remains of the *Combitanic*. The ship shuddered and rolled over, slowly.

"Thanks," Jack croaked. He coughed until he could speak again. "I had the perfect rescue planned. It wasn't supposed to end like this."

Rachel watched as the *Combitanic* disappeared from view. She felt a part of her go with it. Suddenly, she noticed a whole flotilla of lifeboats, filled with grateful survivors, dimly lit by the final flames of the wreck. A quick head count revealed that everyone could have been saved.

"Jack, this is amazing! The ship must have sunk within minutes. How did everyone evacuate so quickly?"

"I've been reading about dynamic network flows," Jack replied. "Rescuing the passengers of the *Combitanic* was just an instance of the quickest transshipment problem. Once I felt the impact of the iceberg, I knew what everyone had to do."

"Oooh, you are so masterful." Rachel melted into Jack's arms. "I want to know your secrets. Please tell me more."

"Well, it's a long story..."

# Chapter 1

# Introduction

Modern society depends on transportation. Goods move from producers to consumers. People commute, shop, and travel. Information speeds through an electronic web connecting homes, schools, and offices around the world. Our lives are filled with decisions about what should go where next and how. For its pervasive role in society, transportation deserves study. Understanding transportation better can lead to greater efficiency — translating into lower costs for transportation providers and better service for their customers.

Interest in transportation spawned the field of *network flows* during the 1940s and 50s. A network consists of *nodes* and *edges*. Each node corresponds to a factory, warehouse, computer, or some other location of production, consumption, or consolidation. Each edge connects a pair of nodes, corresponding to roads, cables, or other channels. Commodities flow from node to node, transported by the edges of the network. Each edge has a capacity that restricts the amount it can transport.

Network flows have many applications other than transportation modeling. Scheduling, personnel assignment, DNA sequencing, and open pit mining are just a tiny sample of the applications that have been posed as network flow problems. For a more complete list, see the textbook of Ahuja *et al* [2].

The study of network flows developed simultaneously with that of linear programming (and optimization in general). Network flow problems are specially structured linear programs and may be solved as such. They also yield to special-purpose combinatorial algorithms, however, and these have received most attention from interested researchers ever since Ford and Fulkerson championed this approach in their landmark text on network flows [16]. The structure of network flow problems leads to solution techniques that are computationally more efficient (both theoretically and practically) than those known to solve linear programs.

The computer science and operations research communities have pursued ever more efficient network flow algorithms for the last four decades. A vast body of literature lies in their wake, but it contains a curious omission. Despite the natural connection between network flows and transportation, most research in network flow theory has ignored the first question asked by any child in the back seat of a car: "Are we there yet?" The parents who endure this persistent line of questioning undoubtedly consider time before deciding where to go next and how.

Ford and Fulkerson were well aware of the importance of time in transportation, and they incorporated it into their network flow model. They generalized the standard definition of a network to include *transit times* between nodes, resulting in a *dynamic network*. Each edge $yz$ in a dynamic network models a pipeline from node $y$ to node $z$. The capacity of edge $yz$ corresponds to the cross-section of the pipeline; it restricts the amount of flow can enter edge $yz$ per unit time. The transit time of edge $yz$ corresponds to the length of the pipeline; it determines how much time passes while flow moves from $y$ to $z$ along edge $yz$. A *dynamic flow* moves over time through a dynamic network; it is an extension of traditional network flow, mapping (edge,time) pairs to flow values, rather than edges to flow values.

Since Ford and Fulkerson, dynamic network flows have been studied extensively by the operations research community. The survey articles of Aronson [4] and

Powell *et al* [47] summarize much of the progress made. Research in this area has focused on both theoretical questions of mathematical modeling and practical concerns of algorithm implementation; it has not, however, paid much attention to algorithmic theory. In particular, very little is known about the computational complexity of dynamic network flow problems.

In this thesis, we study dynamic network flows and focus on the development of theoretically efficient algorithms. Our work bridges a gap between the study of traditional network flows and that of dynamic network flows. Like most researchers of traditional network flows, we strive for the best possible asymptotic bounds on worst-case algorithm performance. However, our algorithms solve dynamic problems based on a more realistic model typically considered by researchers of transportation. Most of our algorithms are the first efficient techniques known for their respective problems. Initial versions of our results appeared in [27] and [28].

**Polynomial vs Pseudopolynomial Algorithms.** Any dynamic network flow algorithm must somehow represent dynamic flow on an edge as that flow changes with time. The standard technique is to consider discrete steps of time and make a copy of the original network for every time step from time zero until the *time horizon* $T$, after which there is no flow left in the network. This process results in a *time-expanded network*. All algorithms based on time-expanded networks have running times depending polynomially on $T$; such algorithms are *pseudopolynomial*.

Most of the algorithms in this thesis have true polynomial running times; they depend polynomially on $\log T$, not on $T$. We achieve this breakthrough by eliminating the time-expanded network. Rather than computing the flow on each edge at every individual time step, our algorithms produce solutions characterized by long time intervals for each edge during which its flow remains constant. We do not explicitly compute these intervals; they are a by-product of *chain-decomposable flows*, a well-structured class of dynamic flows that we define in Chapter 4. All of our algorithms compute chain-decomposable flows.

Our polynomial algorithms may represent not only a theoretical breakthrough but also a technique with practical significance. Dynamic network flows are often used to model continuous-time problems in the real world. A more accurate model relies on finer granularity, implying more time steps before the dynamic flow is finished (with each step representing less real time). The performance of a pseudopolynomial algorithm degrades linearly (or worse) with the improvement of model granularity; this restricts the accuracy that can be achieved by the model. The performance of a polynomial algorithm, however, degrades logarithmically (or better) with finer model granularity, and so may allow much more accurate modeling than traditional techniques.

**Universally Maximum Dynamic Flows.** Ford and Fulkerson [16] focused their attention on the most basic of all dynamic network flow problems, the *maximum dynamic flow problem*. An instance of this problem is a time horizon $T$ and a dynamic network with one source and one sink; a solution is a feasible dynamic flow that sends as much as possible from the source to the sink within time $T$. Ford and Fulkerson discovered an ingenious polynomial algorithm for this problem from which all the algorithms of this thesis descend.

Gale [18], Wilkinson [54], and Minieka [40] considered a variant of the maximum dynamic flow problem in which the sink must receive as much flow as possible by every intermediate time step up to and including $T$. In addition, flow should leave the source as late as possible. We call such a flow a *universally maximum dynamic flow*. The existence of such flows is not obvious, but was proved by Gale [18]. Later, Wilkinson [54] and Minieka [40] independently described pseudopolynomial algorithms to compute these flows.

In Chapter 5, we reexamine the almost identical algorithms of Wilkinson and Minieka and describe an equally similar algorithm based on chain-decomposable flows. Our perspective leads to the first polynomial algorithm to compute the value of a universally maximum dynamic flow on any edge at any single time step (*i.e.*, a time-step snapshot of the full solution). We also describe the first polynomial algorithm to approximate a universally maximum dynamic flow within a factor of $(1+\epsilon)$, for any $\epsilon > 0$. The algorithm runs in polynomial time with respect to $\epsilon^{-1}$ and the input network.

**Quickest Flows.** Another variant of the maximum dynamic flow problem is the *quickest flow problem*, in which we are given a flow value $v$ and must find a feasible dynamic flow that sends $v$ units of flow from the source to the sink within the least possible time. The quickest flow problem can be reduced to the maximum dynamic flow problem by binary search. Burkard *et al* [7] studied this problem and described more efficient solution techniques.

The *evacuation problem* is a multi-source generalization of the quickest flow problem: we are given a supply value for each source and asked to find a feasible dynamic flow that sends the specified amount from each source to the sink in the least overall time. In a traditional network without transit times, the evacuation problem reduces trivially to a single-source flow problem (by using a supersource). In the dynamic setting, however, the evacuation problem is much more complicated than the quickest flow problem. It is more useful also: the evacuation problem was originally formulated as a practical model for building evacuation. The evacuation problem has been studied by several groups [5,8,10,11,26,31]. All of them avoided the bad news that the only technique known to compute optimal solutions required exponential time and space.

In Chapters 6–8 we describe the first (strongly) polynomial algorithm for the *quickest transshipment problem*, a generalization of the evacuation problem which allows multiple sinks in addition to multiple sources. Previously, no polynomial algorithm was known even for the evacuation problem with just two sources. Our algorithm not only solves the quickest transshipment problem but also computes an integral flow when given integral data.

**Network Scheduling.** Although the evacuation and quickest transshipment problems are easy to motivate by disaster contingency plans, they have applications in other settings as well. Consider a set of computers on a network; each machine has some queue of unit-size jobs waiting to execute. A job can either run locally or go into the network for remote execution. Each link of the network is characterized by a capacity and transit time. The problem of executing these jobs so that the last job executes as soon as possible reduces easily to the integral quickest transshipment problem. By applying our quickest transshipment algorithm, we obtain the first polynomial algorithm for scheduling unit-size jobs on a network. Other groups have obtained polynomial algorithms for unit-job network scheduling, but only for very special cases. Deng *et al* [13] considered networks with unit transit times and no capacities. Fizzano and Stein [15] considered ring networks with unit transit times and unit capacities. Our algorithm handles general networks with arbitrary non-negative integer capacities and transit times.

**Outline of Thesis.** We define basic terminology and notation in Chapter 2. In Chapter 3, we formally introduce the dynamic network flow problems of this thesis and survey research related to them. Chapter 4 presents *chain-decomposable flows*; these are a well-structured class of dynamic flows used by all the algorithms of this thesis; they are a generalization of *temporally repeated flows* as described by Ford and Fulkerson [16].

Our algorithmic results begin in Chapter 5, where we discuss universally maximum dynamic flows. In Chapter 6 we consider *lexicographically maximum dynamic flows*; these are a multi-terminal generalization of maximum dynamic flows. Our results culminate in Chapters 7 and 8; we show that lexicographically maximum dynamic flows can be used to solve the *dynamic transshipment problem*, a version of the quickest transshipment problem in which the time horizon is specified; this leads to our quickest transshipment algorithm. We conclude with some observations and open problems in Chapter 9.

# Chapter 2

# Definitions

A *directed graph* is a set of edges $E$ associated with a set of nodes $V$; each edge in $E$ is an ordered pair of nodes $yz$ in $V$. An $(s,t)$-*path* in a directed graph $(V,E)$ is an ordered set of edges $P = (y_0y_1, y_1y_2, y_2y_3, \ldots, y_{k-1}y_k)$ such that each $y_i$ is distinct, $s = y_0$, and $t = y_k$. If $y_ky_0$ is also an edge and $k > 1$, then appending $y_ky_0$ to $P$ creates a *cycle*. A *forest* is a graph that contains no cycle, even when each edge may be used forward or backward. An $(s,t)$-*cut* is a partition of the node set $V$ into two parts, $C$ and $V \setminus C$, such that $C$ contains $s$ but not $t$; we also say "cut" to refer to the set of edges that have one endpoint in $C$ and the other in $V \setminus C$.

A *dynamic network* $\mathcal{N} = (V, E, u, \tau, S)$ consists of a directed graph $(V,E)$ with a non-negative integral[1] *capacity* $u_{yz}$ and integral *transit time* $\tau_{yz}$ associated with each edge $yz$ in $E$, and a special subset of nodes $S$ called *terminals*. The maximum capacity is denoted by $U$ and the maximum transit time by $\mathcal{T}_{\mathrm{m}}$. We also refer to transit times as *length* and *cost*. A terminal is either a *source* or a *sink*; the set of sources is denoted by $S^+$ and the set of sinks by $S^-$. We distinguish edges with positive capacity by including them also in set $E^+$. For simplicity, we make the following assumptions. $E$ is symmetric: if $yz \in E$ then $zy \in E$; and $\tau$ is antisymmetric: if $yz \in E$ then $\tau_{yz} = -\tau_{zy}$. There are no parallel edges or

[1]We restrict ourselves to integral capacities merely for simplicity. We could also allow rational- or real-valued capacities.

zero-length cycles in $E$. $E^+$ contains no opposite edges, and edges in $E^+$ have non-negative transit time. Sources have no entering edges in $E^+$, and sinks have no leaving edges in $E^+$. We use $k$ to denote $|S|$; likewise, $n = |V|$ and $m = |E|$.

Suppose $s$ is a source and $t$ is a sink in network $\mathcal{N}$. A *dynamic* $(s,t)$*-flow* is a function $f_{yz}(\theta)$ on $E \times \mathbb{N}$ that satisfies the following constraints:

$$\forall yz \in E,\ \theta \in \mathbb{N}: \quad f_{zy}(\theta + \tau_{yz}) \quad = \quad -f_{yz}(\theta) \tag{2.1}$$

$$\forall y \in V \setminus \{s,t\},\ \theta \in \mathbb{N}: \quad \sum_{\theta'=0}^{\theta} \sum_{z \in V} f_{yz}(\theta') \quad \leq \quad 0 \tag{2.2}$$

where we use the notation that $f_{yz}(\theta) = 0$ for all $\theta$ if neither $yz$ nor $zy$ is in $E$, and $f_{yz}(\theta) = 0$ for all $yz$ if $\theta < 0$. Equation (2.1) is an *antisymmetry constraint*; inequality (2.2) is a relaxed *conservation constraint* that allows each non-terminal node to store a non-negative amount of *holdover flow* as temporary inventory. (Usually we do not need holdover flow. See the remark at the end of this chapter.) If dynamic flow $f$ is only defined up to some time horizon $T$ (or if it is zero for all time after $T$) then $f$ is a *finite-horizon* dynamic flow. Dynamic flow $f$ with time horizon $T$ is *feasible* if it satisfies *capacity constraints*

$$\forall yz \in E,\ \theta \in \mathbb{N}: \quad f_{yz}(\theta) \quad \leq \quad u_{yz} \tag{2.3}$$

and if there is no holdover flow remaining at non-terminal nodes after time $T$:

$$\forall y \in V \setminus \{s,t\}: \quad \sum_{\theta=0}^{T} \sum_{z \in V} f_{yz}(\theta) \quad = \quad 0. \tag{2.4}$$

In the absence of any time horizon, $f$ is an *infinite-horizon* flow. An infinite-horizon dynamic flow is feasible if it satisfies all capacity constraints. Notice that by the antisymmetry constraint (2.1) for an edge $yz$ in $E^+$, the capacity constraint $f_{zy}(\theta + \tau_{yz}) \leq u_{zy}$ requires that $f_{yz}(\theta) \geq 0$.

Given any time bound $\theta$ and node $y$, the *value* of $f$ at $y$ is the net dynamic flow out of $y$ for all time up to $\theta$:

$$|f_y|_\theta \quad = \quad \sum_{\theta'=0}^{\theta} \sum_{z \in V} f_{yz}(\theta').$$

We usually consider the value of $f$ only at terminals. Without reference to a particular terminal, $|f|_\theta$ is the net dynamic flow into the sink: $-|f_t|_\theta$. (This slightly awkward definition has the advantage that it ignores flow still in transit; the net flow out of the source $|f_s|_\theta$ counts flow still in the network at time $\theta$.) The *throughput* of $f$ on edge $yz$ is the total flow into $yz$ for all time up to $\theta$:

$$|f_{yz}|_\theta = \sum_{\theta'=0}^{\theta} f_{yz}(\theta').$$

In the absence of any time bound, $|f_y|$ and $|f_{yz}|$ are the corresponding infinite sums. Dynamic flows with many sources and sinks are defined analogously. For multi-terminal networks, we denote the net flow out of a subset of terminals $A \subseteq S$ by $|f(A)|_\theta$.

We often associate a multi-terminal network $\mathcal{N}$ with a *supply vector* in $\mathbb{R}^S$ denoted by $v$. We assume that $v_{s_i} \geq 0$ for every source $s_i$ in $S^+$ and $v_{s_i} \leq 0$ for every sink $s_i$ in $S^-$. For any subset of terminals $A \subseteq S$, we denote the total supply of $A$ by $v(A)$. We assume that the total supply of all terminals is zero. We say that supply vector $v$ is *satisfied* by dynamic flow $f$ with horizon $T$ if $|f_{s_i}|_T = v_{s_i}$ for every terminal $s_i$ in $S$.

We refer to flows and circulations in $\mathcal{N}$ in the traditional sense as *static* flows and circulations. A *static* $(s,t)$*-flow* is a function $f_{yz}$ on $E$ that satisfies antisymmetry constraints $f_{yz} = -f_{zy}$ for every edge $yz$, and conservation constraints $\sum_z f_{yz} = 0$ for every node $y \neq s, t$, where we use the notation that $f_{yz} = 0$ if $yz \notin E$. A *static circulation* is defined analogously, except it must satisfy the conservation constraints at all nodes. Static flow or circulation $f$ is feasible if it also satisfies capacity constraints $f_{yz} \leq u_{yz}$ for every edge $yz$. The *residual network* of static flow $f$ subject to capacity $\tilde{u}$ is defined as $\mathcal{N}_{\tilde{u},f} = (V, E, \tilde{u}^f, \tau, S)$, where the *residual capacity* function is $\tilde{u}^f_{yz} = \tilde{u}_{yz} - f_{yz}$. We also use the notation $\mathcal{N}_f$ when the capacities are clear from context. An edge with zero residual capacity is *saturated.* The *value* of a static $(s,t)$-flow $f$ is $|f| = \sum_y f_{yt}$. Multi-source multi-sink static flows are defined analogously.

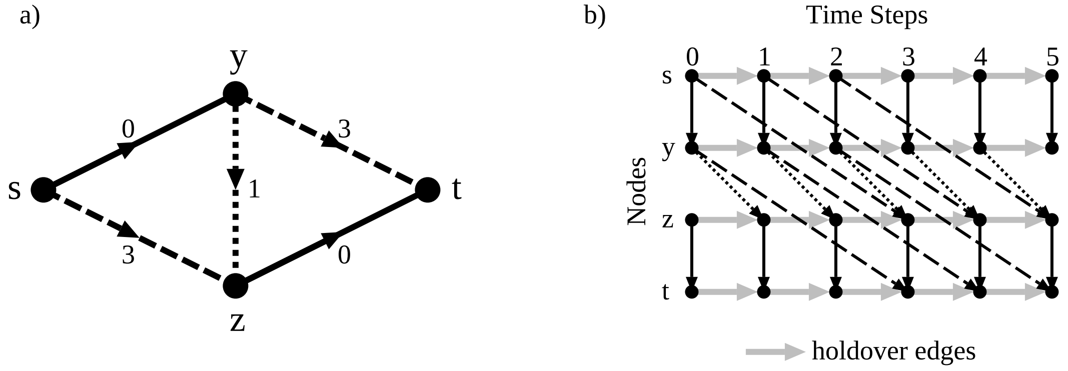


Figure 2.1: Dynamic Network with Transit Times & Time-Expanded Network

A dynamic flow $f$ with time horizon $T$ is equivalent to a static flow in the *time-expanded network* $\mathcal{N}(T) = (V(T), E(T), u^T, \tau^T, S(T))$. Each node $y$ in $V$ has $T+1$ copies in $V(T)$, denoted $y(0), \ldots, y(T)$. Each edge $yz$ in $E$ has $T - |\tau_{yz}| + 1$ copies in $E(T)$, each with capacity $u_{yz}$, denoted $y(\theta)z(\theta + \tau_{yz})$ for any time $\theta$ such that both $y(\theta)$ and $z(\theta + \tau_{yz})$ are in $V(T)$. In addition, $E(T)$ contains a *holdover* edge $y(\theta)y(\theta+1)$ with infinite capacity for each node $y$ and time $0 \leq \theta < T$. An infinite-horizon dynamic flow is equivalent to a static flow in the infinite time-expanded network $\mathcal{N}(*)$, defined analogously to $\mathcal{N}(T)$.

**Example:** Figure 2.1(a) depicts a dynamic network $\mathcal{N}$. Each edge is labeled with its transit time. The capacities are not indicated; we assume unit capacities. Figure 2.1(b) shows the time-expanded network $\mathcal{N}(5)$ associated with this dynamic network. Every edge again has unit capacity, except for the infinite-capacity holdover edges.

Unfortunately, the terminology of dynamic network flows is not consistent throughout the literature. Notably, Orlin [42,43,44] refers to our dynamic networks as static networks, and to our time-expanded networks as dynamic networks.

**Remark on Holdover Flow.** We have defined dynamic flows to allow each node to hold an arbitrary non-negative amount of flow. Conservation constraint (2.2) expresses this capability formally; so do the holdover edges in the time-expanded network. In a real-world network, however, nodes may have very little capacity to hold inventory, and so we may want to tighten constraint (2.2) to equality (or eliminate holdover edges) at non-terminal nodes. Although this is a valid concern, the feasibility of holdover flow does not matter for almost all of the problems described in this thesis: our algorithms solve problems that allow holdover flow but the solutions we find do not actually use holdover flow. There are no exceptions to this rule until we discuss mortal edges in Chapter 8.

# Chapter 3

# Research Survey

## 3.1 Static Network Flows

Traditional static network flows have been studied extensively since Ford and Fulkerson's seminal textbook [16]. The literature on this field is enormous, and most of it is related only tangentially to the results of this thesis. An excellent reference is Ahuja *et al* [2].

We are particularly interested in one topic of static network flow theory: the *minimum-cost flow problem* has important applications to dynamic network flows. This problem requires as input a network $\mathcal{N}$ and a supply vector $v$. A solution is a feasible (static) flow in network $\mathcal{N}$ that satisfies supply $v$ (if such a flow exists) with the additional constraint that $\sum_{yz} \tau_{yz} f_{yz}$ is minimum over all such flows. Ahuja *et al* survey the current state of the art for this much-studied problem in [2].

Most of the algorithms in this thesis solve minimum-cost flow problems repeatedly and so rely on efficient minimum-cost flow algorithms as important subroutines. We are not concerned here with the internals of minimum-cost flow algorithms, however, and so we treat them all as a black box. We use MCF to denote the time of a single minimum-cost flow computation; this time has been bounded within $O(nm \log(n^2/m) \log(n\mathcal{T}_{\mathrm{m}}))$ by Goldberg and Tarjan [21], $O(nm(\log\log U) \log(n\mathcal{T}_{\mathrm{m}}))$ by Ahuja *et al* [1], and $O((m \log n)(m + n \log n))$ by

Orlin [45]. Note that the last algorithm is strongly polynomial.

Our results depend not only on the existence of efficient minimum-cost flow algorithms but also on some basic properties of minimum-cost flows themselves. In the rest of this section, we review properties of minimum-cost flows that are useful in the development of our dynamic flow algorithms; we rely mostly on well-known results of Ford and Fulkerson [16].

**Theorem 3.1.1** (Ford and Fulkerson [16]) Static flow $f$ in network $\mathcal{N}$ is minimum-cost if and only if the residual network $\mathcal{N}_f$ has no negative-cost positive-capacity cycles.

Consider a minimum-cost static flow $f$ in network $\mathcal{N}$. For any nodes $y, z$ in the network, let $d_f(y, z)$ be the length of a shortest $(y, z)$-path in the residual network $\mathcal{N}_f$, or infinity if no such path exists. Note that Theorem 3.1.1 implies that function $d_f$ is well-defined.

**Lemma 3.1.2** (Ford and Fulkerson [16]) If $f$ is a minimum-cost static flow, then for every edge $yz$:

$$\max\{0, d_f(s,z) - \tau_{yz} - d_f(s,y)\}(u_{yz} - f_{yz}) = 0.$$

**Proof:** Suppose $d_f(s,z) - \tau_{yz} - d_f(s,y) > 0$. Then flow $f$ must saturate edge $yz$; otherwise, there is an $(s,z)$-path through $y$ that is shorter than $d_f(s,z)$. ■

**Uniqueness.** Some of our dynamic flow algorithms depend on their minimum-cost flow subroutines to compute a unique minimum-cost flow for each feasible combination of network $\mathcal{N}$ and supply vector $v$. In Chapter 2, we define our networks to have no zero-length cycles; Lemma 3.1.3 implies that any minimum-cost flow algorithm will compute a unique solution given such a network. Lemma 3.1.4 then shows that we lose no generality by assuming networks without zero-length cycles, because we can perturb any cost function lexicographically to guarantee the same property.

**Lemma 3.1.3** Suppose static flow $f$ satisfies supply $v$ in network $\mathcal{N}$. Then $f$ is the unique minimum-cost flow satisfying $v$ in $\mathcal{N}$ if and only if every positive-capacity cycle in the residual network $\mathcal{N}_f$ has positive cost.

**Proof:** Suppose every positive-capacity cycle in $\mathcal{N}_f$ has positive cost; then Theorem 3.1.1 implies that $f$ is a minimum-cost flow. If there were another minimum-cost flow $f' \neq f$ satisfying $v$ in $\mathcal{N}$, then $f' - f$ must consist of zero-cost residual cycles in $\mathcal{N}_f$.

Suppose $f$ is the unique minimum-cost flow satisfying $v$ in $\mathcal{N}$; then Theorem 3.1.1 implies that every positive-capacity cycle in $\mathcal{N}_f$ has non-negative cost. In fact, each of these cycles must have positive cost, or else we can produce another minimum-cost flow different than $f$ by augmenting along a zero-cost residual cycle. ■

**Lemma 3.1.4** Let $\mathcal{N}$ be a network with node set $V$, edge set $E$, and edge cost function $\tau$. Order the edges of $E = \{e_1, \ldots, e_m\}$ and assign each edge $e_i$ a vector cost $\bar{\tau}_{e_i} = (\tau_{e_i}, 2^i)$. For any feasible supply $v$, there is a unique flow $f$ that satisfies supply $v$ for lexicographically minimum cost with respect to $\bar{\tau}$; flow $f$ is also minimum-cost with respect to $\tau$. Furthermore, if network $\mathcal{N}$ has no negative-cost positive-capacity cycles, then the zero flow is the unique minimum-cost circulation in this network.

**Proof:** It is clear that any flow that is lexicographically minimum-cost with respect to $\bar{\tau}$ is also minimum-cost with respect to $\tau$. It is equally clear that there are no zero-length cycles in $\mathcal{N}$ with respect to $\bar{\tau}$. Combined with the assumption that there are no negative-cost positive-capacity cycles in $\mathcal{N}$, the uniqueness of $f$ follows from Lemma 3.1.3. ■

**Monotonicity.** Ford and Fulkerson [16] originally solved the minimum-cost $(s,t)$-flow problem with the *shortest augmenting path algorithm.* Starting with the zero flow, this algorithm repeatedly finds a shortest $(s,t)$-path in the current residual network and then augments the flow along that path. Ford and Fulkerson showed that after each augmentation, the current flow is minimum-cost and the shortest residual path lengths from $s$ and to $t$ do not decrease. The correctness of our dynamic flow algorithms depends crucially on the slightly more general Lemma 3.1.6.

**Lemma 3.1.5** (Ford and Fulkerson [16]) Suppose $f$ is a minimum-cost static flow and nodes $s, t$ are two terminals in network $\mathcal{N}$. Let static flow $g$ augment $f$ along a shortest $(s,t)$-path in residual network $\mathcal{N}_f$. Then (1) $f+g$ is a minimum-cost static flow in network $\mathcal{N}$, and (2) for any node $y$ in the network, $d_f(s,y) \leq d_{(f+g)}(s,y)$ and $d_f(y,t) \leq d_{(f+g)}(y,t)$.

**Lemma 3.1.6** Suppose $f$ is a minimum-cost static flow and nodes $s, t$ are two terminals in network $\mathcal{N}$. Let $g$ be a minimum-cost static $(s,t)$-flow in residual network $\mathcal{N}_f$. Then (1) $f+g$ is a minimum-cost static flow in network $\mathcal{N}$, and (2) for any node $y$ in the network, $d_f(s,y) \leq d_{(f+g)}(s,y)$ and $d_f(y,t) \leq d_{(f+g)}(y,t)$.

**Proof:** Suppose, for a contradiction, that $g$ is a minimum-cost static $(s,t)$-flow in $\mathcal{N}_f$ such that either (1) or (2) is false, and that $|g|$ is minimum over all such flows. (Note that Lemma 3.1.5 implies that $|g| > 1$.) Let $g'$ be the minimum-cost $(|g|-1)$-value flow in $\mathcal{N}_f$; the lemma holds for $g'$. Let unit-value static flow $g''$ augment $f+g'$ along the shortest $(s,t)$-path in residual network $\mathcal{N}_{(f+g')}$. By Lemma 3.1.5, both conditions (1) and (2) hold for static flow $g'+g''$; but since $g'+g''=g$, this contradicts our assumption that condition (1) or (2) is violated. ∎

## 3.2 Dynamic Network Flows

Aronson [4] and Powell *et al* [47] survey the field of dynamic network flows comprehensively. Many results in this area relate only tangentially to this thesis. Below, we survey the results most closely related to our work.

**Maximum Dynamic Flows.** In the *maximum dynamic flow problem*, we are given a time horizon $T$ and a dynamic network $\mathcal{N}$ with a single source $s$ and a single sink $t$; we seek to maximize the value of a feasible dynamic $(s,t)$-flow with time horizon $T$. Ford and Fulkerson [16] focused their efforts in dynamic flows on formulating and solving this problem. They discovered a surprisingly simple polynomial algorithm using *temporally repeated flows*, a well-structured subclass of dynamic flows. We refer to temporally repeated flows as *standard chain-decomposable flows*, a simple type of *chain-decomposable flows*. We define all these terms and discuss the maximum dynamic flow problem more fully in Chapter 4.

**Universally Maximum Dynamic Flows.** Shortly after Ford and Fulkerson solved the maximum dynamic flow problem, Gale [18] considered a more complicated variant. In the *earliest arrival flow problem*, we seek a single feasible dynamic $(s,t)$-flow with specified time horizon $T$ that maximizes the total amount of flow reaching the sink by every time step up to and including $T$. Note that it is not at all clear that such a flow can generally exist. In fact, Gale proved only that such flows do always exist; he conjectured but did not prove an algorithm to compute such flows. Following Gale, Wilkinson [54] and Minieka [40] independently confirmed his conjecture. Minieka also considered the *latest departure flow problem*, in which we seek a feasible dynamic $(s,t)$-flow with time horizon $T$ that maximizes the total amount of flow departing from the source after every time step (subject to the constraint that the flow is finished by time $T$). Minieka showed that a single flow can have both an earliest arrival and a latest departure schedule, and he proved that the very algorithm conjectured by Gale and confirmed by himself and Wilkinson actually solves both the earliest arrival and latest departure flow problems simultaneously. We call such a flow a *universally maximum dynamic flow.* Our terminology conflicts with the traditional nomenclature, in which earliest arrival flows are equivalent to universally maximum dynamic flows, and jointly source- and sink- optimal flows are called earliest arrival latest departure flows. We review and extend Wilkinson's and Minieka's results in Chapter 5.

**Lexicographically Maximum Dynamic Flows.** The *lexicographically maximum dynamic flow problem* is a multi-terminal extension of the maximum dynamic flow problem. We are given a time horizon $T$ and a dynamic network $\mathcal{N}$ with an ordered set of terminals $S = \{s_0, \ldots, s_{k-1}\}$; we seek a feasible dynamic flow that lexicographically maximizes the amount of flow leaving each terminal in the given order. Note that the terminal set includes both sources and sinks. Maximizing the amount of flow leaving a terminal is equivalent to minimizing the amount of flow entering a terminal.

All previous research on this problem has focused on static network flows. Minieka [40] studied lex max static flows with the goal of solving the universally maximum dynamic flow problem, which can be viewed as a special lex max flow problem in a time-expanded network. Minieka discussed only the special case in which all sources are ranked higher than all sinks. Megiddo [37] studied lex max static flows in single-source multi-sink networks, but his problem was somewhat different than ours. Not only did he assume the source to be ranked first, but he also assumed no pre-specified ordering of the sinks. He sought a maximum flow out of the source that would maximize the minimum amount of flow entering any sink, any pair of sinks, etc. Intuitively, a solution to this problem is a maximally balanced flow. Gallo *et al* [19] described a parameterized maximum flow algorithm for which both Minieka's and Megiddo's problems are special cases; the algorithm of Gallo *et al* solves these problems and a host of others in the time required for one preflow-push max-flow computation of Goldberg and Tarjan [20]. All algorithms for lex max static flows apply to lex max dynamic flows via the exponentially large time-expanded network. In Chapter 6, we describe the first polynomial algorithm for the lex max dynamic flow problem.

**Quickest Flows.** Another variant of the maximum dynamic flow problem is to minimize the time horizon given a flow value, rather than maximizing the flow value given a time horizon. In the *quickest flow problem*, we are given a flow amount $v$ and we seek the minimum time $T$ so that there is a feasible dynamic $(s,t)$-flow with time horizon $T$ and value $v$. Burkard *et al* [7] introduced this problem. Once the minimum time $T$ is known, the problem is almost identical to the maximum dynamic flow problem (complicated by the fact that a maximum dynamic flow in time $T$ may have value greater than $v$). The minimum time $T$ can easily be determined by binary search, yielding a simple polynomial algorithm for the quickest flow problem. Burkard *et al* also described more complicated search algorithms with better worst-case bounds than binary search, as well as strongly polynomial algorithms. In contrast, Kagaris *et al* [32] devised a continuous-time version of the problem that is NP-complete.

In the *quickest path problem*, we seek a quickest flow that uses only one path. The quickest path problem can be solved in polynomial time, as discovered by Chen and Chin [9], Rosen *et al* [51], and Hung and Chen [29]. These results were recently improved by Kagaris *et al* [32]. Burkard *et al* [6] also described how to compute the quickest flow using $h$ disjoint paths, for fixed $h$.

The *evacuation problem* is a multi-source single-sink version of the quickest flow problem. Given a dynamic network $\mathcal{N}$ and supply vector $v$, the problem is to find a feasible dynamic flow that satisfies supply $v$ in the minimum overall time, if such a flow exists. Berlin [5] and Chalmet *et al* [8] studied this problem as a means of modeling emergency evacuation from buildings. Jarvis and Ratliff [31] proved that three different optimality criteria can be achieved simultaneously: (1) an earliest arrival schedule that maximizes the total flow into the sink by every time step, (2) overall minimization of the time required to evacuate the network, and (3) minimization of the average time for all flow to reach the sink. Hamacher and Tufecki [26] showed how to prevent unnecessary movement within a building while optimizing evacuation time. They also described how to evacuate a building divided into prioritized zones; a solution must evacuate the highest priority zone as quickly as possible, then the next highest zone as quickly as possible, etc. Choi *et al* [10] studied evacuation given flow-dependent exit capacities; see also [11]. Most of the results above rely on time-expanded networks and so involve pseudopolynomial algorithms; some more efficient techniques are considered but these results fall short of general optimality.

Hajek and Ogier [23] considered a continuous-time version of the evacuation problem in the special case when all transit times are zero. They obtained a polynomial algorithm by reducing this problem to $k$ maximum flow computations. Their solution is a sequence of *stationary dynamic flows*, in which the flow on each edge is the same from one moment of time to the next.

In this thesis, we consider a multi-source and multi-sink version of the quickest flow problem: the *quickest transshipment problem.* We are given a dynamic network $\mathcal{N}$ and supply vector $v$. We seek a feasible dynamic flow with the minimum possible time horizon that satisfies supply $v$, if such a flow exists. The quickest transshipment problem is closely related to the *dynamic transshipment problem*, in which the time horizon $T$ is specified and the goal is to move the appropriate amount of flow through the network within that time, if possible. We describe the first polynomial algorithms for the dynamic transshipment problem in Chapter 7 and for the quickest transshipment problem in Chapter 8.

**Dynamic Dynamic Network Flows.** Although flows in dynamic networks do change with time, we generally assume that dynamic networks themselves do not change with time: edge capacities and transit times are constant. There are some results for networks with time-varying edge characteristics, however. Minieka [41] studied the maximum dynamic flow problem where each edge may be specified with a time when it is added to the network or a time when it is deleted from the network; his model prohibits holdover flow. Halpern [24] further generalized the maximum dynamic flow problem to allow time-varying edge capacities and prohibition of holdover flow on specified nodes during specified time intervals. Although neither Minieka nor Halpern used time-expanded networks, they proved nothing stronger than the correctness and finiteness of their algorithms. In both cases, the advantage over a simple time-expanded approach depends on having only a small list of network changes to manage. In Chapter 8, we give a polynomial algorithm for the dynamic transshipment problem in a network built entirely of *mortal edges*, each of which can admit flow only during its own specified time interval; our solution uses holdover flow.

**Network Scheduling.** In the *unit-size-job network scheduling problem*, we are given a set of computers on a network; each machine has some queue of unit-size jobs waiting to execute; a job can either run locally or go into the network for remote execution. Each link of the network is characterized by a capacity and transit time. The objective is to schedule job execution so that the last job finishes as soon as possible. Deng *et al* [13] described a polynomial algorithm for the special case of this problem when the links have unit transit times and no capacities. Their algorithm uses a time-expanded network, but they obtained a polynomial bound on the time horizon by reducing problems with (exponentially) many jobs to problems with a small (polynomial) number of jobs. Fizzano and Stein [15] considered the very special case of this problem given ring networks with unit transit times and unit capacities; they obtained a particularly efficient polynomial algorithm. In Chapter 8, we reduce the general unit-size-job network scheduling problem to the quickest transshipment problem and so obtain the first polynomial algorithm for the general problem.

Even the general unit-size-job network scheduling problem is a special case in the field of network scheduling. There are many more complicated problems in this area; they are all closely related to well-studied problems in parallel machine scheduling. The difference between network scheduling and parallel machine scheduling is that parallel machine scheduling assumes instantaneous communication between different machines, while network scheduling allows links between machines to be restricted by transit times and/or capacities. Thus, every network scheduling problem is a generalization of some parallel machine scheduling problem (of which there are many, see Lawler *et al* [36]).

Scheduling becomes much more challenging when the jobs are not all unit size. Phillips *et al* [46] studied the problem of scheduling jobs on a network of identical machines connected by uncapacitated links with arbitrary transit times; an optimal schedule completes the last job as soon as possible. They gave a polynomial algorithm to compute a schedule that is no more than twice as long as an optimal schedule, and they proved that this ratio cannot be improved to less than 4/3 unless P=NP. Other network scheduling problems, hardness results, approximation algorithms, and computational results are discussed in the Ph.D. thesis of Veltman [53].

**Infinite-Horizon Dynamic Flows.** All of the dynamic network flow problems we have surveyed so far have finite time horizons. Finite-horizon optimization problems are harder in some sense than infinite-horizon optimization problems. As long as an infinite-horizon dynamic flow is asymptotically optimal, any finite interval of time steps can be suboptimal and the overall flow is still optimal. An optimal finite-horizon dynamic flow, however, cannot be sub-optimal for a single time step. In many finite-horizon dynamic flow problems, the most complicated time steps to schedule are the first few and the last few; the intervening time can often be described by a stationary dynamic flow. In the case of an infinite-horizon dynamic flow problem, the solution has no end and the beginning is inconsequential; thus, stationary flows can often solve infinite-horizon problems. Because stationary flows are so well-structured, they can often be computed by efficient static network flow algorithms.

Orlin formulated and solved two infinite-horizon dynamic flow problems. In

[42] he discussed the *maximum-throughput dynamic flow problem.* In this paper, the *throughput* of a flow at time $\theta$ is the total amount of flow in transit at time $\theta$ — flow that has started in some edge by time $\theta$ but has not reached the end of the edge yet. Orlin showed that there is always a dynamic flow with maximum throughput that is stationary, and that such a stationary flow can be computed via one minimum-cost (static) network flow problem. He also showed how his result is a generalization of Ford and Fulkerson's maximum dynamic flow algorithm.

In a further generalization, Orlin [43] considered the *minimum convex cost dynamic network flow problem.* In this problem, each edge has a convex cost function $c$ in addition to a capacity $u$ and transit time $\tau$. The goal is to find a feasible infinite-horizon dynamic flow $f$ with minimum average cost per time step and zero throughput (using throughput in the same sense as [42]). Here again, Orlin proved that the problem can be solved by a stationary flow, albeit a fractional one; he also showed how to round the fractional stationary solution into a periodic integral one.

**Minimum-Cost Dynamic Flows.** In the finite-horizon case, adding an edge cost function to a dynamic network quickly leads to NP-hard problems. Klinz and Woeginger [34] proved hardness results for some minimum-cost dynamic flow problems and discussed a greedy algorithm that works in some special cases. Hamacher [25] discussed efficient algorithms for minimum-cost dynamic flow problems in a flawed paper that nevertheless motivated some of the important ideas of Chapter 4.

**Continuous-Time Flows.** Continuous time is a natural extension of our dynamic network flow model, and has been considered by a number of researchers. Except for Kagaris *et al* [32], however, all continuous-time models we know of assume that all transit times are zero; the result is like a static network flow that evolves continuously over time. Hajek and Ogier [23] studied the evacuation problem in this setting; they obtained a polynomial algorithm that reduces the problem to $k$ maximum (static) flow problems. More generally, Anderson and Philpott [3] considered a minimum-cost dynamic network flow problem and obtained a continuous zero-transit-time dynamic network version of the traditional network simplex algorithm. Even more generally, Pullan [48] studied *continuous linear programs*, in which finite collections of variables and constraints are replaced by relations between bounded, measurable, and/or continuous functions; these are also called *infinite linear programs*. Pullan described a class of continuous linear programs for which a strong duality result holds and also described approximation and improvement step algorithms for these problems. Our survey has only touched upon continuous linear programming; we do not attempt an exhaustive survey ourselves because of the tangential nature of this topic to our thesis.

**Periodic Networks.** Time-expanded networks are a one-dimensional subclass of periodic networks. A *$k$-dimensional periodic network* is the natural expansion of a dynamic network with $k$-dimensional vector transit times. The general study of periodic networks is not particularly relevant to dynamic network flows; however, these two areas do share a common concern for finding algorithms that are efficient with respect to a small implicit representation of a large but repetitive network. Orlin [44] studied connectivity and colorability properties of infinite time-expanded networks $\mathcal{N}(*)$, giving polynomial algorithms to test these properties using the underlying dynamic network $\mathcal{N}$. Iwano and Steiglitz [30], Kosaraju and Sullivan [35], and Cohen and Megiddo [12] all studied algorithms to detect cycles in periodic networks.

# Chapter 4

# Chain-Decomposable Flows

The dynamic network flow problems in this thesis are equivalent to easy static flow problems in time-expanded networks. However, because the size of a time-expanded network $\mathcal{N}(T)$ depends exponentially on $\log T$, a time-expanded network takes exponentially more space than its underlying dynamic network and time horizon $(\mathcal{N}, T)$ when $T$ is large. Thus, time-expanded networks do not generally lead to efficient algorithms for dynamic network flow problems.

Ford and Fulkerson [16] introduced *temporally repeated flows* to represent some repetitive dynamic flows efficiently. Here we introduce *chain-decomposable flows* to allow efficient representation of a considerably larger class of dynamic flows. We refer to temporally repeated flows as *standard chain-decomposable flows*; they are a simple subclass of chain-decomposable flows. All the algorithms of this thesis compute chain-decomposable flows.

## 4.1 Standard Chain-Decomposable Flows

A *chain flow* $\gamma = \langle v, P \rangle$ is a static flow of positive value $v$ along path (or cycle) $P$ in a network $\mathcal{N} = (V, E, u, \tau, S)$. We denote the length of the chain flow by $\tau(\gamma)$; it is equal to the total length of path $P$. If $P$ includes node $y$, then $\tau_y(\gamma)$ is the length of $P$ from its first node to node $y$. If path $P$ includes edge $yz$, then we say that $\gamma$ *uses* $yz$. Given time horizon $T$ no less than $\tau(\gamma)$, any chain flow $\gamma$ induces a dynamic flow by sending $v$ units of flow along path $P$ every time step from time zero till time $T - \tau(\gamma)$. The last $v$ units of flow finally reach the end of $P$ at time $T$.

A chain flow is *proper* if it starts and ends in the terminal set $S$. Let $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ be a set of proper chain flows. We say that $\Gamma$ is a *chain decomposition* of static flow $f$ if $\sum_{i=1}^{k} \gamma_i = f$, and that $\Gamma$ is a *standard chain decomposition* of $f$ if all chain flows in $\Gamma$ use edges in the same direction as $f$ does. If $\Gamma$ is a standard chain decomposition of $f$, every chain flow in $\Gamma$ is no longer than time horizon $T$, and $f$ is feasible, then $\Gamma$ induces a feasible dynamic flow, obtained by summing the dynamic flows induced by each chain flow in $\Gamma$. A dynamic flow computed in this manner is called a *standard chain-decomposable flow*, and we denote it by $[\Gamma]^T$. In the absence of any time horizon, $\Gamma$ induces an infinite-horizon standard chain-decomposable flow $[\Gamma]$ by repeating each chain flow endlessly.

We denote by $\Gamma_{yz}$ the subset of chain flows in $\Gamma$ that use directed edge $yz$. We say that chain flow $\gamma$ in $\Gamma$ *touches* edge $yz$ if $\gamma \in \Gamma_{yz} \cup \Gamma_{zy}$, and that the $\gamma$-induced flow $[\{\gamma\}]^T$ *covers* $yz$ at time $\theta$ if $[\{\gamma\}]^T_{yz}(\theta) \neq 0$. In the infinite-horizon case, note that if $[\{\gamma\}]$ covers $yz$ at time $\theta$, then $[\{\gamma\}]$ also covers $yz$ at all times after $\theta$.

**Example:** Consider the network of Figure 2.1(a) with time horizon $T = 5$. Figure 4.1(a) shows a maximum static flow in this network, decomposed into two chain flows. Figures 4.1(b,c) show the dynamic flow induced by these chain flows — first in a time-expanded network and then in a sequence of time-step snapshots.

## 4.2 Maximum Dynamic Flows

Standard chain-decomposable flows are a very simple class of dynamic flows. Interesting dynamic flow problems cannot necessarily be solved by standard chain-decomposable flows. Ford and Fulkerson [16] showed, however, that standard

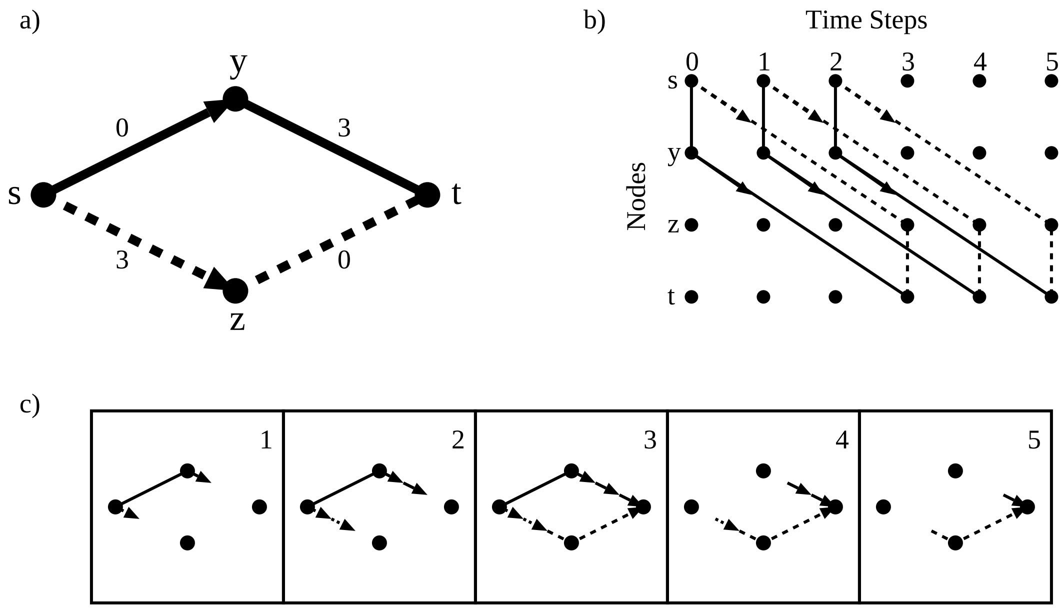


Figure 4.1: Standard Chain-Decomposable Flow

chain-decomposable flows do suffice to solve the maximum dynamic flow problem. We review their results in this section.

Consider a standard chain decomposition $\Gamma$ of static flow $f$. Ford and Fulkerson observed that the value of $[\Gamma]^T$ depends only on $f$ and is independent of the choice of standard chain decomposition $\Gamma$.

**Lemma 4.2.1** (Ford and Fulkerson [16]) Suppose $\Gamma$ is a standard chain decomposition of flow $f$, and that every chain flow in $\Gamma$ is no longer than $T$. Then

$$|[\Gamma]^T| = (T+1)|f| \;-\; \sum_{yz \in E^+} \tau_{yz} f_{yz}.$$

**Proof:** Each chain flow $\gamma$ in $\Gamma$ contributes $|\gamma|$ units of flow from time 0 till time $T - \tau(\gamma)$; therefore, the total dynamic flow value is

$$\sum_{\gamma \in \Gamma} (T + 1 - \tau(\gamma))\, |\gamma|$$

which easily reduces to the desired formula. ■

Lemma 4.2.1 suggests an algorithm to find a standard chain-decomposable flow of maximum value. Given a dynamic network $\mathcal{N}$ with source $s$, sink $t$, and time horizon $T$, add an artificial return edge $ts$ to the network with infinite capacity and length $-(T+1)$. Let $f$ be a minimum-cost (static) circulation in this network. Notice that $f$ maximizes the formula of Lemma 4.2.1, so that any standard chain decomposition of $f$ induces a maximum standard chain-decomposable flow.

We generalize the above construction to multi-terminal networks as follows: Let $\mathcal{N}$ be a dynamic network with source set $S^+$, sink set $S^-$, and time horizon $T$. Add a superterminal $\psi$ to the network, connected to each source $s_i$ by an infinite-capacity zero-transit-time edge $\psi s_i$ and connected to each sink $s_i$ by an infinite-capacity edge $s_i\psi$ with transit time $-(T+1)$. We call the result an *extended network*, and we denote it by $\mathcal{N}^T$. When discussing an extended network $\mathcal{N}^T$, we implicitly include the artificial edges in edge set $E$ (or $E^+$); if we wish to exclude artificial edges, we use the notation $E \setminus \psi$ (or $E^+ \setminus \psi$).

We have just proved the following corollary to Lemma 4.2.1:

**Corollary 4.2.2** (Ford and Fulkerson [16]) Finding a maximum standard chain-decomposable flow in dynamic network $\mathcal{N}$ with time horizon $T$ reduces to finding a minimum-cost (static) circulation in extended network $\mathcal{N}^T$.

The following theorem shows that standard chain-decomposable flows are sufficient to solve the general maximum dynamic flow problem. This result crowns Ford and Fulkerson's work on dynamic flows:

**Theorem 4.2.3** (Ford and Fulkerson [16]) For any dynamic network and time horizon, a maximum standard chain-decomposable flow is also a maximum dynamic flow.

**Proof:** Given network $\mathcal{N}$ and time horizon $T$, let $f$ be a minimum-cost circulation in extended network $\mathcal{N}^T$, and $\Gamma$ be a standard chain decomposition of $f$. We assume that $f$ is non-zero, since otherwise it is clear that the maximum dynamic flow is zero and the theorem is an easy consequence of Theorem 3.1.1. Consider the cut $C = \{y(\theta) : \theta \geq d_f(s, y)\}$ in infinite time-expanded network $\mathcal{N}(*)$. It is easy to see that $C$ separates source $s(0)$ from sink $t(T)$ and that $[\Gamma]^T$ is a feasible dynamic flow with time horizon $T$; we show that $[\Gamma]^T$ saturates cut $C$ by deriving the capacity of $C$ and comparing it to Lemma 4.2.1. This proves the theorem.

Suppose edge $y(\theta)z(\theta')$ crosses $C$; then $y(\theta) \in C$ and $z(\theta') \notin C$. (The other direction is considered as opposite edge $z(\theta')y(\theta)$.) The definition of $C$ implies that $y(\theta)z(\theta')$ is either a non-holdover edge with capacity $u_{yz}$ or a reverse holdover edge with zero capacity. If $y(\theta)z(\theta')$ is not a holdover edge, then $\theta' = \theta + \tau_{yz}$ and $y(\theta)z(\theta')$ crosses $C$ for every $\theta : d_f(s,y) \leq \theta < d_f(s,z) - \tau_{yz}$. Thus, the total capacity of cut $C$ is

$$\sum_{yz \in E^+} \max\{0, d_f(s,z) - \tau_{yz} - d_f(s,y)\} u_{yz}.$$

By applying Lemma 3.1.2 to edges $yz$ and $zy$, the above formula reduces to

$$\sum_{yz \in E^+} (d_f(s,z) - d_f(s,y)) f_{yz} - \sum_{yz \in E^+} \tau_{yz} f_{yz}.$$

Finally, note that $d_f(s,s) = 0$ and $d_f(s,t) = T + 1$. (The second equality depends on our assumption that $f$ is non-zero.) These facts plus flow conservation imply that the capacity of cut $C$ is $(T+1)|f| - \sum_{yz \in E^+} \tau_{yz} f_{yz}$, so that Lemma 4.2.1 completes the theorem. ■

## 4.3 Chain-Decomposable Flows

Ford and Fulkerson used standard chain-decomposable flows to reduce the maximum dynamic flow problem in network $\mathcal{N}$ to a simple static flow problem in extended network $\mathcal{N}^T$. One would hope for similar results in other dynamic flow problems; unfortunately, standard chain-decomposable flows are not enough. Just about every problem except maximum dynamic flow fails to yield to Ford and Fulkerson's efficient technique. Most research in this area resorts to time-expanded networks. Notable exceptions to this rule are the infinite-horizon dynamic flow problems of Orlin [42,43] and zero-transit-time evacuation problem of Hajek and

Ogier [23] (which use stationary flows), and the quickest flow problem of Burkard *et al* [7] (which uses standard chain-decomposable flows).

Here we introduce chain-decomposable flows. Chain-decomposable flows extend the foundations of standard chain-decomposable flows only slightly; but the resulting extension supports considerably more range to solve interesting dynamic flow problems that previously were handled only with time-expanded networks. Our extension relies on a generalized notion of chain decomposition. Recall that every chain flow in a standard chain decomposition of flow $f$ must use edges in the same direction as $f$ does. The chain flows in a non-standard chain decomposition may use oppositely directed flows on edges. Chain-decomposable flows are induced by non-standard chain decompositions in exactly the same manner as standard chain-decomposable flows are induced by standard chain decompositions, but our generalization does introduce some complications.

**Time Travel.** A chain flow in a non-standard chain decomposition may use a residual edge with negative transit time. Luckily, network flow is simpler than most things one can imagine going back in time, and this simplicity allows some intuition to explain how the time travel works. Consider an edge $yz$ with positive transit time $\tau_{yz}$; symmetry implies that there is also an edge $zy$ with negative transit time $-\tau_{yz}$. A unit of flow sent from $y$ at time $\theta$ reaches $z$ at time $\theta + \tau_{yz}$; on the other hand, one negative unit of flow sent from $z$ at time $\theta + \tau_{yz}$ reaches $y$ at time $\theta$. These are two different ways of describing exactly the same event. In both cases $y$ loses one unit of flow at time $\theta$ and $z$ gains a unit of flow at time $\theta + \tau_{yz}$, and the residual capacity of edge $yz$ behaves the same way from either perspective. Now suppose one unit of flow leaves $z$ at time $\theta$; it travels back in time and reaches $y$ at time $\theta - \tau_{yz}$. This backward time travel is equivalent to sending one negative unit of flow from $y$ at time $\theta - \tau_{yz}$; it reaches $z$ at time $\theta$. Figures 4.2(b1-b3) show the equivalence of positive flow moving backward in time and negative flow moving forward in time.

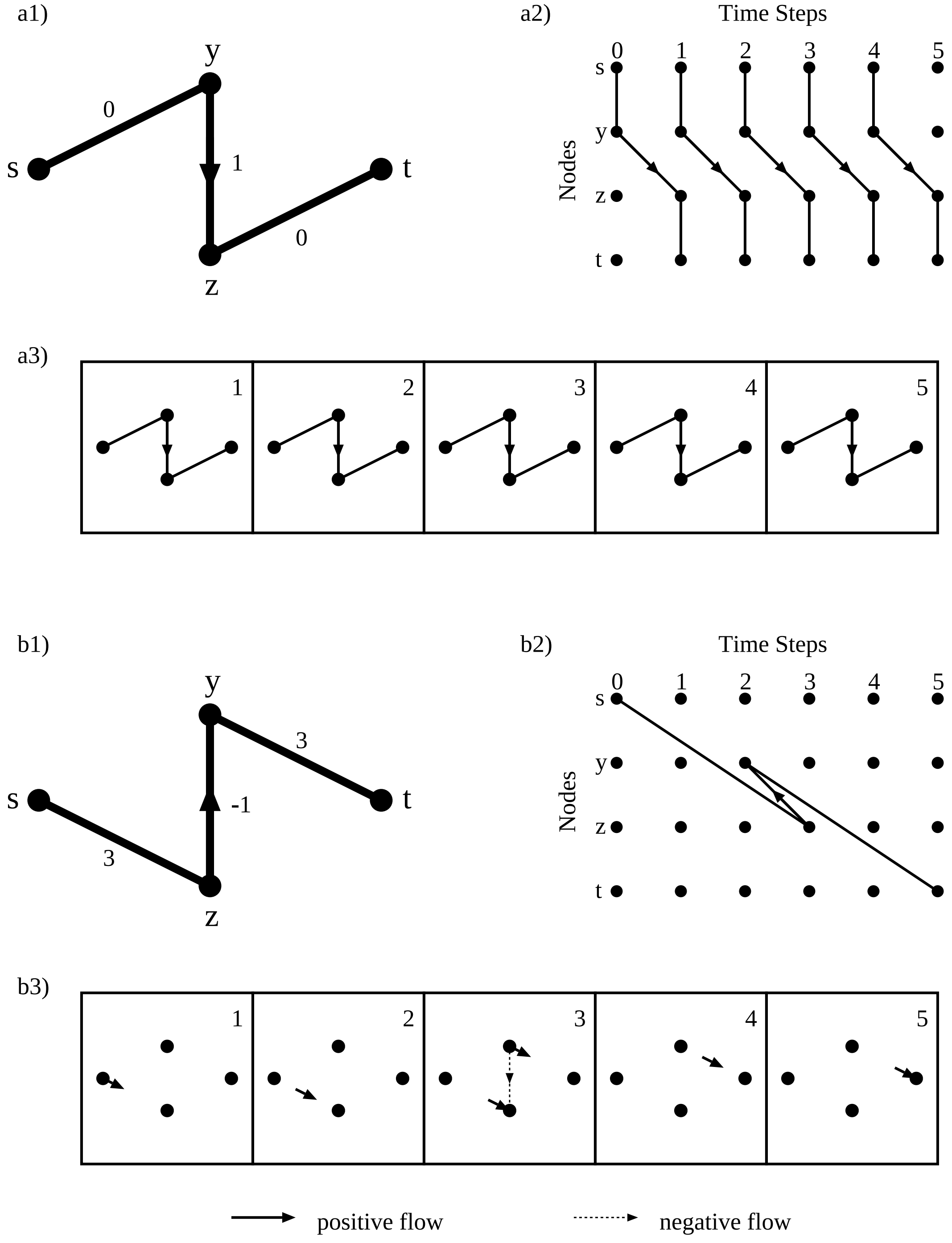


Figure 4.2: Chain Flows and Induced Dynamic Flows

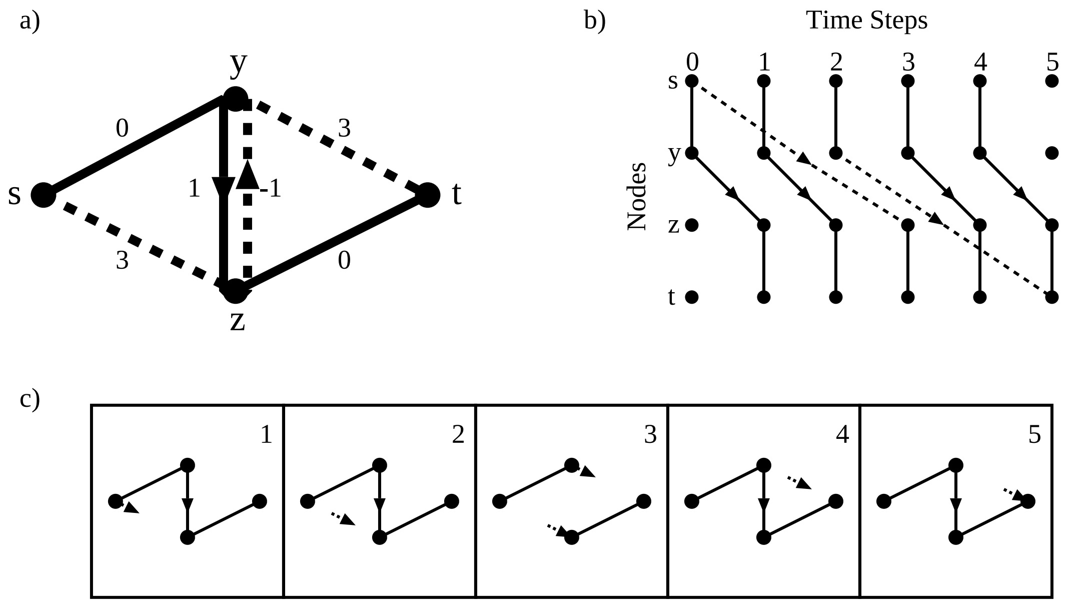


Figure 4.3: Non-Standard Chain-Decomposable Flow

**Example:** Consider the network of Figure 2.1(a) with time horizon $T = 5$. Figure 4.3(a) shows a non-standard chain decomposition of a maximum static flow in this network. The dynamic flows induced separately by each of these chain flows are depicted in Figures 4.2(a1-b3). Added together, this non-standard chain decomposition induces the dynamic flow of Figures 4.3(b,c).

**Feasibility.** Feasibility of chain-decomposable flows depends crucially on the timing of when each chain flow brings dynamic flow to each edge. Consider the previous example depicted in Figure 4.3; let $\gamma_1$ denote the solid-edged chain flow and $\gamma_2$ the dashed chain flow. Chain flow $\gamma_2$ uses opposite edge $zy$ to cancel the flow on edge $yz$ induced by $\gamma_1$. If the transit times of the network were different so that $\gamma_2$ reached $y$ before $\gamma_1$, however, then the flow induced by $\gamma_2$ would start cancelling on $yz$ before $\gamma_1$ provided any flow there to cancel. The resulting dynamic flow would be infeasible, even though $\gamma_1 + \gamma_2$ is feasible. Thus, in contrast to the trivial feasibility proofs of standard chain-decomposable flows, we have to argue carefully to show that our chain-decomposable flows do not violate capacity constraints.

**Finite vs Infinite Horizons.** When we introduced standard chain-decomposable flows, we distinguished between finite-horizon flows of the form $[\Gamma]^T$ and infinite-horizon flows of the form $[\Gamma]$. In the context of general chain-decomposable flows, however, this distinction is not as clear as it first appears.

Suppose $\Gamma$ is a chain decomposition of flow $f$ in extended network $\mathcal{N}^T$. Let $\gamma^*$ be the longest chain flow in $\Gamma$. Notice that for any time $\theta$ after $\tau(\gamma^*)$, static flow $f$ exactly characterizes dynamic flow $[\Gamma]$, that is $[\Gamma]_{yz}(\theta) = f_{yz}$. Suppose also that $f$ is the zero flow. If $\Gamma$ is a standard chain decomposition of $f$, then the dynamic flow $[\Gamma]$ is trivially zero. If we allow non-standard chain decompositions, however, then $[\Gamma]$ may be non-zero. Even though $[\Gamma]$ is non-zero, however, the fact that $f$ is zero means that $[\Gamma]$ will be zero by time $\tau(\gamma^*)$ and will remain zero forever after. In other words, $[\Gamma]$ may be a finite-horizon dynamic flow.

We use non-standard chain decompositions of the zero flow in the extended network $\mathcal{N}^T$ to induce dynamic flows. Each chain flow starts and ends at the superterminal $\psi$. This is our most general framework for chain-decomposable flows; every problem solved by standard chain-decomposable flows and finite-horizon chain-decomposable flows $[\Gamma]^T$ can also be solved using this method.

**Example:** Suppose network $\mathcal{N}$ consists of a single edge $st$ of unit length and unit capacity. Consider the extended network $\mathcal{N}^T$ with $T = 1$. Figure 4.4 depicts this network with two different chain flows. Chain flow $\gamma_1$ pushes one unit of flow along cycle $(\psi s, st, t\psi)$; chain flow $\gamma_2$ pushes one unit of flow along the opposite cycle $(\psi t, ts, s\psi)$. Taken together, $\{\gamma_1, \gamma_2\}$ is a chain decomposition of the zero flow, and $[\{\gamma_1, \gamma_2\}]$ is a finite-horizon dynamic flow with time horizon $T$. Notice that $[\{\gamma_1, \gamma_2\}]$ is equivalent to the standard chain-decomposable flow $[\{\gamma\}]^T$, where $\gamma$ is a unit-value chain flow on path $(st)$.

We conclude this chapter by deriving the edge throughput of dynamic flow $[\Gamma]$. We then apply this result in Corollary 4.3.2 to obtain the value of such a flow into

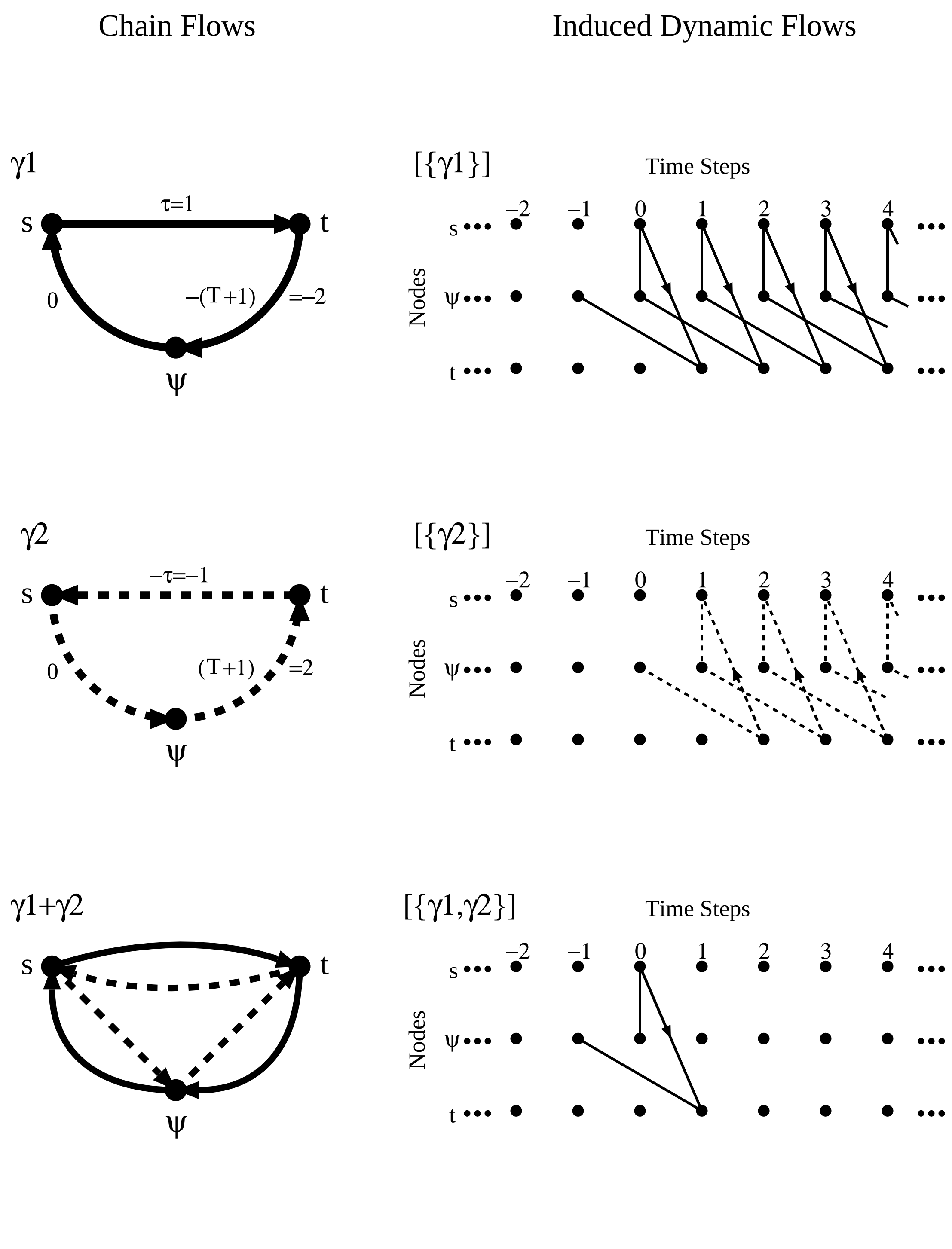


Figure 4.4: Zero-Flow Chain Decomposition

and out of each terminal in the network; we show an equivalence between dynamic flow value and static flow cost.

**Lemma 4.3.1** Suppose $\Gamma$ is a chain decomposition of the zero flow in extended network $\mathcal{N}^T$. For any edge $yz$, the total throughput of $[\Gamma]$ on edge $yz$ is

$$|[\Gamma]_{yz}| = \sum_{\gamma\in\Gamma_{zy}} \tau_y(\gamma)|\gamma| - \sum_{\gamma\in\Gamma_{yz}} \tau_y(\gamma)|\gamma|.$$

**Proof:** Consider time $\theta$, late enough so that every chain flow in $\Gamma$ that touches edge $yz$ covers $yz$ by time $\theta$. The throughput up to time $\theta$ on edge $yz$ is equal to the amount of flow from $\Gamma_{yz}$ that reaches $y$ by time $\theta$ minus the amount of flow from $\Gamma_{zy}$ that reaches $y$ by time $\theta$:

$$\begin{aligned} |[\Gamma]_{yz}| &= \sum_{\gamma\in\Gamma_{yz}} (\theta - \tau_y(\gamma))|\gamma| - \sum_{\gamma\in\Gamma_{zy}} (\theta - \tau_y(\gamma))|\gamma| \\ &= \theta\left(\sum_{\gamma\in\Gamma_{yz}} |\gamma| - \sum_{\gamma\in\Gamma_{zy}} |\gamma|\right) - \sum_{\gamma\in\Gamma_{yz}} \tau_y(\gamma)|\gamma| + \sum_{\gamma\in\Gamma_{zy}} \tau_y(\gamma)|\gamma|. \end{aligned}$$

Because $\Gamma$ is a chain decomposition of the zero flow, the $\theta$ term cancels itself out and the lemma follows. ∎

**Corollary 4.3.2** Suppose $\Gamma$ is a chain decomposition of the zero flow in extended network $\mathcal{N}^T$. For any source $s$ in $S^+$, the total value of $[\Gamma]$ out of $s$ is

$$|[\Gamma]_{\psi s}| = \sum_{\gamma\in\Gamma_{s\psi}} \tau(\gamma)|\gamma| = \sum_{yz\in E^+} \left(\tau_{yz} \sum_{\gamma\in\Gamma_{s\psi}} \gamma_{yz}\right).$$

For any sink $t$ in $S^-$, the total value of $[\Gamma]$ into $t$ is

$$\begin{aligned} |[\Gamma]_{t\psi}| &= \sum_{\gamma\in\Gamma_{\psi t}} (T+1)|\gamma| - \sum_{\gamma\in\Gamma_{t\psi}} (\tau(\gamma)+T+1)|\gamma| \\ &= -\sum_{yz\in E^+} \left(\tau_{yz} \sum_{\gamma\in\Gamma_{t\psi}} \gamma_{yz}\right). \end{aligned}$$

In the case of a single-source single-sink network, Corollary 4.3.2 is equivalent to Lemma 4.2.1. Notice that Corollary 4.3.2 includes the artificial edges of the extended network in $E^+$, but Lemma 4.2.1 does not and so adds that cost explicitly.

# Chapter 5

# Universally Maximum Dynamic Flows

We defined universally maximum dynamic flows in Chapter 3. Wilkinson [54] and Minieka [40] described virtually identical algorithms to compute universally maximum dynamic flows. Both algorithms are pseudopolynomial; neither uses any form of chain-decomposable flows. Wilkinson did consider chain-decomposable flows, but only to prove that standard chain-decomposable flows are insufficient to solve the universally maximum dynamic flow problem. In the next section, however, we solve the problem with (general) chain-decomposable flows; our algorithm is little more than a restatement of Wilkinson and Minieka's results in terms of chain-decomposable flows.

The perspective of our algorithm leads us to some new results in this area. In the next section, we describe the first polynomial algorithm to compute a *time-step snapshot* of a universally maximum dynamic flow. Rather than computing a complete dynamic flow $f$, this algorithm computes the dynamic flow value on a particular edge $yz$ at a particular time step $\theta$ (*i.e.*, $f_{yz}(\theta)$). In the final section of this chapter, we consider approximating a universally maximum dynamic flow. We describe a simple yet novel scaling algorithm that approximates a universally

```
Γ ← ∅
f ← zero flow
while d_f(s,t) ≤ T {
    P ← shortest (s,t)-path in N_f
    v ← minimum residual capacity of P
    augment f by v along P
    Γ ← Γ + {⟨v,P⟩}
}
return Γ
```

Figure 5.1: Universally Maximum Dynamic Flow Algorithm

maximum dynamic flow within a factor of $(1+\epsilon)$, for any $\epsilon > 0$. The algorithm runs in polynomial time with respect to $\epsilon^{-1}$ and the input network.

## 5.1 Exact Algorithms

Wilkinson [54] and Minieka [40] solved the universally maximum dynamic flow problem with an approach based on Ford and Fulkerson's shortest augmenting path algorithm [16]. We present essentially the same algorithm, but we express the solution as a chain-decomposable flow. The algorithm is shown in Figure 5.1.

Consider a dynamic network $\mathcal{N}$ with time horizon $T$. Suppose that a static minimum-cost maximum flow $f^*$ in $\mathcal{N}$ is computed via shortest augmenting paths. Let $\gamma_1, \gamma_2, \ldots, \gamma_l$ be the sequence of augmentations, $\Gamma^i = \{\gamma_1, \ldots, \gamma_i\}$, and $f^i = \sum_{j=1}^{i} \gamma_j$. Then $\Gamma^i$ is a chain decomposition of $f^i$. Lemma 3.1.5 implies that $\tau(\gamma_i) \leq \tau(\gamma_{i+1})$, and so there is some $k : 0 \leq k \leq l$ such that $\Gamma^k$ is the set of all chain flows $\gamma_i$ no longer than $T$. The algorithm of Figure 5.1 computes $\Gamma^k$. The following three theorems show that $[\Gamma^k]^T$ is a universally maximum dynamic flow.

**Theorem 5.1.1** (Wilkinson [54] and Minieka [40]) $[\Gamma^k]^T$ is a feasible dynamic flow.

**Proof:** The flow conservation constraints are trivial, and so we consider only capacity constraints. $[\Gamma^0]^T$ is zero and trivially obeys all capacity constraints. Suppose $0 < i \leq k$ and $[\Gamma^{i-1}]^T$ is feasible; we show that $[\Gamma^i]^T$ is feasible. Note that $[\Gamma^i]^T = [\Gamma^{i-1}]^T + [\{\gamma_i\}]^T$. Consider the capacity constraint of edge $yz$ at time $\theta$. If $[\{\gamma_i\}]^T_{yz}(\theta) = 0$, then the induction hypothesis implies that this constraint is not violated. Otherwise $[\{\gamma_i\}]^T_{yz}(\theta) \neq 0$, which means that $d_{f^{i-1}}(s,y) \leq \theta \leq T - d_{f^{i-1}}(y,t)$. Lemma 3.1.5 implies that the distances $d_{f^{i-1}}(s,y)$ and $d_{f^{i-1}}(y,t)$ increase monotonically with $i$, so that any for any $j < i$, if $\gamma_j$ touches $yz$ then $\gamma_j$ covers $yz$ at time $\theta$. This means $[\Gamma^{i-1}]^T_{yz}(\theta) = f^{i-1}_{yz}$. Since $[\{\gamma_i\}]^T$ is a feasible dynamic flow in residual network $\mathcal{N}_{f^{i-1}}$, the capacity constraint is not violated. ■

The following theorem implies that an earliest arrival flow with symmetric departure and arrival schedules is also a latest departure flow, and hence a universally maximum dynamic flow.

**Theorem 5.1.2** (Minieka [40]) Consider dynamic network $\mathcal{N}$ with time horizon $T$. If an earliest arrival flow in $\mathcal{N}$ sends $v_\theta$ units of flow into the sink at each time step $\theta$, then a latest departure flow in $\mathcal{N}$ sends $v_\theta$ units of flow out of the source at each time step $T - \theta$.

**Proof:** For any time $\theta$, an earliest arrival flow maximizes the total amount of dynamic flow reaching the sink by time $\theta$. Similarly, a latest departure flow maximizes the total amount of dynamic flow leaving the source from time $T - \theta$ till time $T$. These are both maximum dynamic flows in $\theta$ time steps, and hence have the same value. Since these cumulative values are equal for any time bound $\theta$, the amounts arriving at the sink at any individual time step $\theta$ and departing from the source at any time step $T - \theta$ must also be identical. ■

**Theorem 5.1.3** (Wilkinson [54] and Minieka [40]) $[\Gamma^k]^T$ is a universally maximum dynamic flow.

**Proof:** We first prove that $[\Gamma^k]^T$ is an earliest arrival flow. For any time $\theta \leq T$, let $\Gamma^k_\theta = \{\gamma_i \in \Gamma^k : \tau(\gamma_i) \leq \theta\}$. Lemma 3.1.5 implies that the sum of chain flows in $\Gamma^k_\theta$ is a minimum-cost flow; in fact, it can be converted to a minimum-cost circulation by adding a return edge $ts$ to the network with length $-(\theta + 1)$. Thus, by Corollary 4.2.2 and Theorem 4.2.3, $\Gamma^k_\theta$ induces a maximum dynamic flow with time horizon $\theta$. Note also that $\Gamma^k - \Gamma^k_\theta$ has no effect on the flow value until after time $\theta$.

Finally, we prove that $[\Gamma^k]^T$ is a latest departure flow and hence a universally maximum dynamic flow. Consider the dynamic flow induced by any single chain flow $\gamma_i$ in $\Gamma^k$. The arrival and departure of flow obey a simple symmetry: for every $|\gamma_i|$ units of flow arriving at the sink at time $\theta$, there are $|\gamma_i|$ units of flow departing from the source at time $T - \theta$. Summing over all chain flows in $\Gamma^k$, we observe that the departure schedule of $[\Gamma^k]^T$ is symmetric to the arrival schedule, and so Theorem 5.1.2 implies that $[\Gamma^k]^T$ is a latest departure flow. ■

**Snapshot Algorithm.** Zadeh [55] constructed pathologically bad networks for the shortest augmenting path algorithm, proving that our universally maximum dynamic flow algorithm can generate an exponential number of chain flows. Thus, regardless of whether we compute universally maximum dynamic flows in the context of time-expanded networks or chain-decomposable flows, we do not yet have a polynomial algorithm. However, by considering chain flows rather than time-expanded networks, we can derive a polynomial algorithm to compute the universally maximum dynamic flow value on a specified edge $yz$ at a specified time $\theta$. We present such an algorithm in Figure 5.2.

Our snapshot algorithm relies on the insight that, given edge $yz$ and time $\theta$, we no longer need to know every chain flow in $\Gamma^k$ that induces a universally maximum dynamic flow; instead, we need only know the sum of all chain flows in $\Gamma^k$ that cover edge $yz$ at time $\theta$. This sum of chain flows is a minimum-cost static flow that we can compute efficiently, as shown in the following theorem:

```
f ← zero flow
if d_f(s,y) > θ or d_f(y,t) > T − θ then return 0 else {
    L ← 0
    f ← minimum-cost maximum flow
    if d_f(s,y) ≤ θ and d_f(y,t) ≤ T − θ then return f_yz else {
        H ← |f|
        while (H − L > 1) {
            f ← minimum-cost ⌈(H + L)/2⌉-value flow
            if d_f(s,y) > θ or d_f(y,t) > T − θ then H ← |f| else L ← |f|
        }
        f ← minimum-cost H-value flow
        return f_yz
    }
}
```

Figure 5.2: Snapshot of Universally Maximum Flow on Edge $yz$ at Time $\theta$

**Theorem 5.1.4** Let $f^*$ be the universally maximum dynamic flow for network $\mathcal{N}$ with time horizon $T$. For any edge $yz$ and time step $\theta$, we can compute $f^*_{yz}(\theta)$ in $O(\log(nU)\,\mathrm{MCF}\,)$ time.

**Proof:** From Theorem 5.1.3 and Lemma 3.1.4, we can consider a unique universally maximum dynamic flow $f^* = [\Gamma^k]^T$. Let

$$\Gamma^k(y,\theta) = \{\gamma_{i+1} \in \Gamma^k : d_{f^i}(s,y) \leq \theta \text{ and } d_{f^i}(y,t) \leq T - \theta\}.$$

Then the desired flow value $f^*_{yz}(\theta)$ is equal to $\sum_{\gamma\in\Gamma^k(y,\theta)} \gamma_{yz}$. Lemma 3.1.5 implies that the distances $d_{f^i}(s,y)$ and $d_{f^i}(y,t)$ increase monotonically with $i$, so that $\Gamma^k(y,\theta)$ is a chain decomposition of a minimum-cost flow. Note that $f^*_{yz}(\theta)$ depends only on this minimum-cost flow and not on the chain decomposition.

The algorithm of Figure 5.2 computes a minimum-cost flow $f$ equal to the sum of chain flows in $\Gamma^k(y,\theta)$. Lower bound $L$ maintains the property that $\Gamma^k(y,\theta)$ contains the shortest augmentation that can be added to an $L$-value minimum-cost flow; by Lemma 3.1.5, this also means that $\Gamma^k(y,\theta)$ contains every chain flow used when computing an $L$-value minimum-cost flow by shortest augmenting paths. Upper bound $H$ maintains the property that no augmentation to an $H$-value minimum-cost flow can be in $\Gamma^k(y,\theta)$. The algorithm terminates when $H = L+1$ (or degenerately when $H = 0$ or $H = L+1 =$the maximum flow value), at which point an $H$-value minimum-cost flow $f$ satisfies $f^*_{yz}(\theta) = f_{yz}$.

The running time of the algorithm is easy to bound. The gap $H - L$ is initially no more than $nU$. Each iteration cuts the gap in half and is dominated by one minimum-cost flow computation. ■

## 5.2 Approximation Algorithm

In this section we develop an efficient capacity-scaling algorithm that approximates a universally maximum dynamic flow within a factor of $(1+\epsilon)$, for any $\epsilon > 0$. The algorithm has the unusual feature of scaling upwards. Traditional scaling

$\Gamma \leftarrow \emptyset;\quad \Delta \leftarrow 1;\quad \tilde{u} \leftarrow u;\quad f \leftarrow$ zero flow

while ($\exists$ $(s,t)$-path in $\mathcal{N}_{\tilde{u},f}$ of length $\leq T$) {

  $\sigma \leftarrow 0$

  while ( ($\sigma < m\Delta/\epsilon$) and

      ($\exists$ $(s,t)$-path in $\mathcal{N}_{\tilde{u},f}$ of length $\leq T$) ) {

    $P \leftarrow$ shortest $(s,t)$-path in $\mathcal{N}_{\tilde{u},f}$

    $v \leftarrow$ minimum residual capacity of $P$

    augment $f$ by $v$ along $P$

    $\Gamma \leftarrow \Gamma + \{\langle v, P\rangle\}$

    $\sigma \leftarrow \sigma + v$

  }

  $\Delta \leftarrow 2\Delta$

  $\forall yz \in E:\ \tilde{u}_{yz} \leftarrow \tilde{u}_{yz} - (\tilde{u}^f_{yz} \bmod \Delta)$

}

return $\Gamma$

Figure 5.3: Universally Maximum Flow $(1+\epsilon)$-Approximation Algorithm

algorithms work initially with capacities rounded by a big scaling factor; the idea is that large capacity edges are more important than small capacity edges. In a dynamic flow, however, a small capacity edge that is short might carry more flow than a large capacity edge that is long. Our algorithm is based on the idea that short chain flows are more important than long ones.

The algorithm is shown in Figure 5.3. In essence, it computes a minimum-cost flow via shortest augmenting paths in a repeatedly rounded network. We use the chain decomposition defined by the sequence of augmentations to induce a dynamic flow. The rounding guarantees that the number of augmentations can be bounded by a polynomial in $n$, $\log U$, and $\epsilon^{-1}$.

The algorithm starts by augmenting along exact shortest paths. Depending on $\epsilon$, the algorithm periodically rounds down the edge capacities according to an increasing scaling factor $\Delta$. This rounding implies that all residual capacities are integer multiples of $\Delta$, so that subsequent augmentations carry at least $\Delta$ units of flow in the static network. Notice that if an edge $yz$ in $E^+$ carries less than $\Delta$ units of flow, then rounding down the residual capacity of $zy$ results in a negative capacity for edge $zy$, *i.e.*, a lower bound on the flow of edge $yz$. This corresponds to "irreversible" flow on edge $yz$.

We need some additional notation in order to analyze the algorithm. Say there are $k+1$ scaling phases during the algorithm, numbered 0 to $k$. We index phases so that $\Delta = 2^i$ during the inner loop of phase $i$. Let $\Gamma^{-1} = \emptyset$ and $\Gamma^i$ denote the set of chain flows at the end of phase $i$; let $T^i$ denote the length of the longest chain flow in $\Gamma^i$; let $f^i$ denote the static flow after phase $i$; and let $\tilde{u}^i$ denote the rounded capacity function at the *beginning* of phase $i$.

**Theorem 5.2.1** $[\Gamma^k]^T$ is a feasible dynamic flow.

**Proof:** By induction on the number of scaling phases. The first scaling phase is identical to the exact algorithm of Wilkinson and Minieka, and so $\Gamma^0$ induces a feasible dynamic flow by Theorem 5.1.1. Suppose that $\Gamma^i$ induces a feasible dynamic flow and $0 \le i < k$. The next scaling phase starts with rounded capacity $\tilde{u}^{i+1}$. By Theorem 5.1.1 we know that $\Gamma^{i+1} - \Gamma^i$ induces a feasible dynamic flow in the residual network $\mathcal{N}_{\tilde{u}^{i+1}, f^i}$, but we need to show that $\Gamma^{i+1}$ induces a feasible dynamic flow in the original network $\mathcal{N}$. This feasibility rests primarily on the fact that the rounding process does not increase the capacity of any edge. One consequence of this fact is that $\Gamma^{i+1} - \Gamma^i$ induces a feasible dynamic flow in residual network $\mathcal{N}_{u,f^i}$. Another consequence of this fact is that shortest residual $(s,y)$- and $(y,t)$-path lengths cannot decrease between scaling phases, for every node $y$. This monotonicity implies that if some chain flow in $\Gamma^{i+1} - \Gamma^i$ covers an edge $yz$

at time $\theta$, then every chain flow in $\Gamma^i$ that touches $yz$ must also cover it at time $\theta$, so that $[\Gamma^i]_{yz}(\theta) = f^i_{yz}$. Combined with the first consequence noted above, this proves the theorem. ■

Because our capacity-scaling algorithm can reduce edge capacities between scaling phases, the maximum dynamic flow value in the rounded network may be less than the original maximum dynamic flow value. Next, we analyze this lost dynamic flow value. For notational simplicity, we use the observation that for any feasible $[\Gamma]^T$ and time $\theta : 0 \leq \theta \leq T$, the value $|[\Gamma]^T|_\theta$ is equal to $|[\Gamma]|_\theta$.

First we consider the lost dynamic flow value due to a single rounding. Let $\Lambda^i_*$ denote a chain decomposition inducing an earliest arrival flow in the residual network $\mathcal{N}_{\tilde{u}^i, f^i}$, and let $\tilde{\Lambda}^i$ denote a chain decomposition inducing an earliest arrival flow in the further rounded residual network $\mathcal{N}_{\tilde{u}^{i+1}, f^i}$. For any time $\theta$, the loss in dynamic flow value due to the rounding at the end of phase $i$ is $|[\Lambda^i_*]|_\theta - |[\tilde{\Lambda}^i]|_\theta$. The following lemma bounds this loss by $\epsilon$ times the value of the dynamic flow induced by the chain flows added during phase $i$.

**Lemma 5.2.2** If $0 \leq i < k$ and $0 \leq \theta \leq T$ then $|[\Lambda^i_*]|_\theta - |[\tilde{\Lambda}^i]|_\theta \ \leq \ \epsilon|[\Gamma^i - \Gamma^{i-1}]|_\theta$.

**Proof:** Let static flow $f^i_*$ equal the sum of all chain flows in $\Lambda^i_*$. Construct $\hat{f}^i$ from $f^i_*$ by repeatedly finding any edge where $f^i_*$ violates the rounded capacity constraint $\tilde{u}^{i+1}$, and then subtracting some $2^i$-value chain flow in $f^i_*$ that uses that edge. Because the rounding of phase $i$ reduces the capacity of any edge by at most $2^i$, this process subtracts no more than $m$ chain flows from $f^i_*$. Furthermore, every canceled chain flow has length at least $T^i$, since after phase $i$ there is no $(s,t)$-path of length less than $T^i$ in the residual network $\mathcal{N}_{\tilde{u}^i, f^i}$. Let $\hat{\Lambda}^i$ denote a standard chain decomposition of the resulting flow $\hat{f}^i$. We have that

$$|[\Lambda^i_*]|_\theta - |[\hat{\Lambda}^i]|_\theta \ \leq \ m2^i(\theta - T^i + 1).$$

Note that $[\hat{\Lambda}^i]^T$ is feasible for the earliest arrival flow problem defined on the rounded residual network $\mathcal{N}_{\tilde{u}^{i+1}, f^i}$, but $[\tilde{\Lambda}^i]^T$ is feasible and optimal for the same problem, so that $|[\tilde{\Lambda}^i]|_\theta \geq |[\hat{\Lambda}^i]|_\theta$, and therefore

$$|[\Lambda^i_*]|_\theta - |[\tilde{\Lambda}^i]|_\theta \;\leq\; m2^i(\theta - T^i + 1).$$

Finally, $0 \leq i < k$ implies that the chain flows of $\Gamma^i - \Gamma^{i-1}$ have total value at least $m2^i/\epsilon$. Since each of these chain flows has length at most $T^i$, we obtain

$$|[\Gamma^i - \Gamma^{i-1}]|_\theta \;\geq\; (m2^i/\epsilon)(\theta - T^i + 1).$$

■

**Theorem 5.2.3** Suppose $0 \leq \theta \leq T$. Let $v^*_\theta$ denote the maximum dynamic flow value in time $\theta$. The algorithm of Figure 5.3 computes dynamic flow $[\Gamma^k]^T$ in time $O(m\epsilon^{-1}(m + n\log n)\log U)$ such that $v^*_\theta \leq (1+\epsilon)|[\Gamma^k]^T|_\theta$.

**Proof:** The claimed running time follows easily. There there are $O(\log U)$ scaling phases. Capacity rounding guarantees that there are $O(m/\epsilon)$ augmentations per phase, each of which requires one non-negative edge length shortest path computation with running time $O(m + n\log n)$ (see Fredman and Tarjan [17]).

We use Lemma 5.2.2 to prove approximate optimality. Theorem 5.1.3 implies that phase $i+1$ begins to compute an earliest arrival flow in the rounded residual network $\mathcal{N}_{\tilde{u}^{i+1}, f^i}$ of phase $i$. This means that $|[(\Gamma^{i+1} - \Gamma^i) + \Lambda^{i+1}_*]|_\theta = |[\tilde{\Lambda}^i]|_\theta$. Similarly, we have $v^*_\theta = |[\Gamma^0 + \Lambda^0_*]|_\theta$. Now we get the following chain of equalities:

$$\begin{aligned}
&|[\Gamma^0]|_\theta + |[\Lambda^0_*]|_\theta \\
&\quad= |[\Gamma^k]|_\theta + \sum_{i=0}^{k-1}\Big(|[\Gamma^i]|_\theta - |[\Gamma^{i+1}]|_\theta\Big) + |[\Lambda^0_*]|_\theta \\
&\quad= |[\Gamma^k]|_\theta + \sum_{i=0}^{k-1}\Big(|[\Lambda^{i+1}_*]|_\theta - |[\tilde{\Lambda}^i]|_\theta\Big) + |[\Lambda^0_*]|_\theta \\
&\quad= |[\Gamma^k]|_\theta + \sum_{i=0}^{k-1}\Big(|[\Lambda^i_*]|_\theta - |[\tilde{\Lambda}^i]|_\theta\Big) + |[\Lambda^k_*]|_\theta.
\end{aligned}$$

Since the phase-$k$ residual network $\mathcal{N}_{\tilde{u}^k, f^k}$ contains no $(s,t)$-paths of length less than $\theta$, it follows that $|[\Lambda^k_*]|_\theta = 0$. Applying Lemma 5.2.2 and observing $|[\Gamma^{k-1}]|_\theta \leq |[\Gamma^k]|_\theta$, we obtain $|[\Gamma^0 + \Lambda^0_*]|_\theta \;\leq\; |[\Gamma^k]|_\theta + \epsilon|[\Gamma^k]|_\theta$. ■

**Corollary 5.2.4** $[\Gamma^k]^T$ is a $(1+\epsilon)$-approximate universally maximum dynamic flow.

**Proof:** Theorem 5.2.3 shows that $[\Gamma^k]^T$ is $(1+\epsilon)$-approximate earliest arrival flow. The approximate optimality of the departure schedule follows from the symmetry of optimal arrival and departure schedules (see Theorem 5.1.2) and the fact that for every unit of flow in $[\Gamma^k]^T$ arriving at the sink at time $\theta$, there is a corresponding unit of flow departing from the source at time $T-\theta$ (see the proof of Theorem 5.1.3).

■

# Chapter 6

# Lexicographically Maximum Dynamic Flows

In the *lexicographically maximum dynamic flow problem*, we are given a dynamic network $\mathcal{N}$ with time horizon $T$ and an ordering of the terminals $S = \{s_0, \ldots, s_{k-1}\}$; we seek a feasible dynamic flow that lexicographically maximizes the amount of flow leaving each terminal in the given order. Note that the terminal set includes both sources and sinks. Maximizing the amount of flow leaving a terminal is equivalent to minimizing the amount of flow entering a terminal. Let $S_i$ denote the high-priority subset of terminals $\{s_0, \ldots, s_{i-1}\}$ and $S_0 = \emptyset$. We denote an instance of the lex max dynamic flow problem by the triplet $(\mathcal{N}, \mathcal{C}, T)$, where $\mathcal{C}$ is the chain of nested high-priority subsets $\{S_0, \ldots, S_k\}$.

We describe a polynomial algorithm to compute lex max dynamic flows. Our lex max dynamic flow algorithm is an important subroutine in our algorithms for solving dynamic transshipment and quickest transshipment problems, which we describe in subsequent chapters.

## 6.1 Applications of Lex Max Flows

Lex max flows are especially useful and interesting because of one important property, which we repeat from Minieka [40] and Megiddo [37]. Consider a dynamic network $\mathcal{N}$ with time horizon $T$. For any subset of terminals $A \subseteq S$, let $o(A)$ denote the maximum amount of flow that the sources in $A$ can send jointly to the sinks in $S \setminus A$ in time $T$ without regard to the needs of other terminals. (Notice that we can compute $o(A)$ as a single-source single-sink maximum dynamic flow value from a supersource connected to the sources of $A$ to a supersink connected to the sinks outside of $A$.) Minieka and Megiddo both observed the following fundamental property of multi-terminal flows:

**Lemma 6.1.1** (Minieka [40] and Megiddo [37]) Suppose dynamic flow $f$ solves the lex max dynamic flow problem $(\mathcal{N}, \mathcal{C}, T)$. Then $|f(S_i)|_T = o(S_i)$ for every high-priority subset $S_i \in \mathcal{C}$.

In other words, a lexicographically maximum flow (static or dynamic) *simultaneously* sends as much flow as possible out of *every* high-priority subset of terminals, for any network and any ordering of that network's terminals.

Our interest in the lex max dynamic flow problem stems primarily from its applications to the dynamic transshipment problem. In the *dynamic transshipment problem*, we are given a dynamic network $\mathcal{N}$ with time horizon $T$ and supply vector $v$. We seek a feasible dynamic flow with time horizon $T$ that satisfies supply $v$, if such a flow exists. We denote an instance of the dynamic transshipment problem by the triplet $(\mathcal{N}, v, T)$. In the next chapter, we show how to reduce the dynamic transshipment problem to the lex max dynamic flow problem.

In this section, we describe a simpler application of lex max dynamic flows. Given a dynamic network $\mathcal{N}$ with time horizon $T$, let $\mathcal{P}$ be the set of all supply vectors $v$ for which the dynamic transshipment problem $(\mathcal{N}, v, T)$ is feasible. Optimizing any linear objective function over set $\mathcal{P}$ reduces to solving a lex max dynamic flow problem:

**Theorem 6.1.2** Given a dynamic network $\mathcal{N}$ with time horizon $T$, let $\mathcal{P}$ be the set of all supply vectors $v \in \mathbb{R}^S$ for which the dynamic transshipment problem $(\mathcal{N}, v, T)$ is feasible. For any cost vector $c \in \mathbb{R}^S$, we can find $v^* \in \mathcal{P}$ such that $\forall v \in \mathcal{P} : c^T v \leq c^T v^*$ via one lex max dynamic flow computation.

**Proof:** Suppose that $c_{s_0} \geq c_{s_1} \geq \ldots \geq c_{s_{k-1}}$. If this is not the case, reindex the terminals so that it is; the time required to sort the terminals is dominated by the time required to compute a lex max dynamic flow. For any supply vector $v$, observe that its cost is

$$c^T v = c_{s_{k-1}} v(S_k) + \sum_{i=1}^{k-1} (c_{s_{i-1}} - c_{s_i}) v(S_i).$$

Compute lex max dynamic flow $f$. Let $v^*_{s_i} = |f_{s_i}|_T$ for every terminal $s_i$ in $S$. By Lemma 6.1.1, $v^*(S_i)$ is maximal over all $v$ in $\mathcal{P}$ for every high-priority subset $S_i$. The theorem then follows from the facts that $v(S_k) = 0$ for all $v$ in $\mathcal{P}$ and $c_{s_0} \geq c_{s_1} \geq \ldots \geq c_{s_{k-1}}$. ■

We revisit Theorem 6.1.2 in the next chapter, describing it in terms of submodular functions and extended polymatroids.

## 6.2 Lex Max Dynamic Flow Algorithm

Minieka [40] described an intuitive algorithm for computing a lex max static flow in network $\mathcal{N}$. Start by computing a maximum (static) flow jointly from $S_{k-1}$ to $s_{k-1}$. In the residual network of this flow, compute a maximum flow jointly from $S_{k-2}$ to $s_{k-2}$. Continue in this fashion until the final iteration sends flow from $s_0$ to $s_1$ through the residual network of all previous iterations. Each iteration of this algorithm saturates another $(S_i, S \setminus S_i)$-cut and so ultimately leads to a lexicographically maximal flow.

Our algorithm for the dynamic version of this problem has a similar structure. We use $k$ iterations of a maximum flow computation in an evolving residual network; however, each iteration computes a maximum dynamic flow rather than a maximum static flow. We induce maximum dynamic flows with standard chain decompositions of minimum-cost static flows, a technique of Ford and Fulkerson [16] presented in Chapter 4. Beginning with the zero flow and an empty chain decomposition $\Gamma$, our lex max dynamic flow algorithm computes successive layers of minimum-cost static flows in the residual networks of previous layers. Each layer yields a standard chain decomposition that is added to set $\Gamma$. At the end of the algorithm, $[\Gamma]$ is a lexicographically maximum dynamic flow.

We present our lex max dynamic flow algorithm formally in Figure 6.1. It starts by adding a superterminal $\psi$ to the input network and connecting $\psi$ to each source $s_i$ in $S^+$ by an infinite-capacity zero-transit-time artificial edge $\psi s_i$. Let $\mathcal{N}^k$ denote the resulting network, $f^k$ the zero flow in this network, and $\Gamma^k$ the empty set.

The algorithm consists of $k$ iterations indexed in descending fashion. In iteration $i = k-1, \ldots, 0$ we consider terminal $s_i$. If $s_i$ is a source, then we delete edge $\psi s_i$ from network $\mathcal{N}^{i+1}$ to create $\mathcal{N}^i$, and we compute a minimum-cost maximum $(\psi, s_i)$-flow $g^i$ in the residual network of flow $f^{i+1}$ in network $\mathcal{N}^i$. If $s_i$ is a sink, we add an infinite-capacity edge $s_i\psi$ with transit time $-(T+1)$ to network $\mathcal{N}^{i+1}$ to create network $\mathcal{N}^i$, and we compute a minimum-cost circulation $g^i$ in the residual network of flow $f^{i+1}$ in network $\mathcal{N}^i$. To complete each iteration (for a source or sink), we add $g^i$ to $f^{i+1}$ to obtain flow $f^i$, compute a standard chain decomposition $\Delta^i$ of flow $g^i$, and add $\Delta^i$ to $\Gamma^{i+1}$ to get chain decomposition $\Gamma^i$.

**Theorem 6.2.1** For any lex max dynamic flow problem $(\mathcal{N}, \mathcal{C}, T)$, a lex max dynamic flow can be computed in $O(k\,\mathrm{MCF})$ time.

**Proof:** The algorithm consists of $k$ iterations, each of which is dominated by one minimum-cost flow computation. The correctness of the algorithm is proved in the next section. ■

$V \leftarrow V \cup \{\psi\}$

$E \leftarrow E \cup \{\psi s_i : s_i \in S^+\}$ with $u_{\psi s_i} = \infty$ and $\tau_{\psi s_i} = 0$

$f^k \leftarrow$ zero flow

$\Gamma^k \leftarrow \emptyset$

for $i \leftarrow k-1, \ldots, 0$ {

    if $s_i \in S^+$ {

        delete edge $\psi s_i$ from $E$

        $g^i \leftarrow$ minimum-cost maximum $(\psi, s_i)$-flow in $\mathcal{N}^i_{f^{i+1}}$

    }

    else {

        add edge $s_i\psi$ to $E$ with $u_{s_i\psi} = \infty$ and $\tau_{s_i\psi} = -(T+1)$

        $g^i \leftarrow$ minimum-cost circulation in $\mathcal{N}^i_{f^{i+1}}$

    }

    $f^i \leftarrow f^{i+1} + g^i$

    $\Delta^i \leftarrow$ standard chain decomposition of $g^i$

    $\Gamma^i \leftarrow \Gamma^{i+1} + \Delta^i$

}

return $\Gamma^0$

Figure 6.1: Lex Max Dynamic Flow Algorithm

## 6.3 Proof of Correctness

This section culminates with three theorems, proving that $[\Gamma^0]$ is a feasible dynamic flow with time horizon $T$ that maximizes the amount of flow leaving every high-priority subset $S_i$. First, however, we prove several preliminary lemmas.

**Lemma 6.3.1** For any iteration $i$, static flow $f^i$ is a minimum-cost circulation in network $\mathcal{N}^i$.

**Proof:** By induction on the number of iterations. Initially, $f^k$ (the zero flow) is a minimum-cost circulation in network $\mathcal{N}^k$ because the input network $\mathcal{N}$ has no negative-cost residual cycles, and hence neither does $\mathcal{N}^k$.

Suppose that the lemma is true for iteration $i+1$ and that $s_i$ is a source. Lemma 3.1.6 implies that $f^i$ is a minimum-cost flow. We need only worry about disturbing the balance of flow at nodes $\psi$ and $s_i$. The artificial edges in $\mathcal{N}^i$ connecting sink nodes to superterminal $\psi$ and the fact that the sources of input network $\mathcal{N}$ have no incoming capacity guarantee that a maximum $(\psi, s_i)$-flow in $\mathcal{N}^i_{f^{i+1}}$ is exactly equal in value to flow $f^{i+1}$ on edge $\psi s_i$. Thus, deleting edge $\psi s_i$ and then adding a maximum $(\psi, s_i)$-flow leaves both $\psi$ and $s_i$ balanced with zero net flow.

The remaining case is trivial. If $s_i$ is a sink and the lemma is true for iteration $i+1$, then Lemma 3.1.6 implies that $f^i$ is a minimum-cost circulation in $\mathcal{N}^i$. ■

For any node $y$ and iteration $i$, let $p^i(y)$ denote the length of a shortest $(\psi, y)$-path in the residual network of flow $f^i$ in network $\mathcal{N}^i$ (or infinity if no such path exists). Note that Lemma 6.3.1 and Theorem 3.1.1 imply that there are no negative-cost cycles in this residual network, so that $p^i(y)$ is well-defined.

**Lemma 6.3.2** For any node $y$ and iteration $i$, $p^i(y) \geq p^{i+1}(y)$.

**Proof:** Consider iteration $i$ and its effect on shortest residual path lengths. Suppose terminal $s_i$ is a source. Shortest path lengths cannot decrease after deleting edge $\psi s_i$, obviously; neither can they decrease after adding a minimum-cost $(\psi, s_i)$-flow, because of Lemma 3.1.6.

If $s_i$ is a sink, iteration $i$ first adds edge $s_i\psi$ to the network and then computes a minimum-cost circulation $g^i$ in the resulting residual network. Adding an edge can decrease shortest residual path lengths only if the new shortest paths use the new edge. We are concerned only with paths from $\psi$ to the rest of the network, and new edge $s_i\psi$ can reduce these path lengths only by creating a negative-cost residual cycle through node $\psi$. However, adding the minimum-cost circulation $g^i$ not only saturates all negative-cost residual cycles but also cannot decrease any shortest residual path lengths, because of Theorem 3.1.1 and Lemma 3.1.6, respectively. ■

**Lemma 6.3.3** Suppose $\gamma \in \Delta^{i-1}$ and $y, z \in V$. Then $\gamma$ does not cover $yz$ at any time before $p^i(y)$.

**Proof:** By definition, residual network $\mathcal{N}^i_{fi}$ contains no $(\psi, y)$-path of length less than $p^i(y)$. We claim that chain flows in $\Delta^{i-1}$ are likewise constrained. If $s_{i-1}$ is a source, then the path of any chain flow in $\Delta^{i-1}$ is a path in $\mathcal{N}^i_{fi}$. If $s_{i-1}$ is a sink, then every edge but $s_{i-1}\psi$ of each chain flow is in $\mathcal{N}^i_{fi}$; and since $s_{i-1}\psi$ is the last edge of each chain flow, it has no effect on when any chain flow covers any other edge. ■

**Lemma 6.3.4** Suppose $\gamma \in \Delta^i$ and $y, z \in V$. If $\gamma$ touches $yz$, then $\gamma$ covers $yz$ at time $p^i(y)$.

**Proof:** Suppose, for a contradiction, that $\gamma$ touches $yz$ but does not cover it at time $p^i(y)$. Then $\gamma$ gives a residual $(y, \psi)$-path in $\mathcal{N}^i_{fi}$ of cost less than $-p^i(y)$. Together with the definition of $p^i(y)$, this means that network $\mathcal{N}^i_{fi}$ has a negative-cost cycle, contradicting Lemma 6.3.1 and Theorem 3.1.1. ■

We are now ready for the main theorems, which prove the correctness of our lex max dynamic flow algorithm.

**Theorem 6.3.5** $[\Gamma^0]$ is a feasible dynamic flow.

**Proof:** First we consider capacity constraints, proving feasibility by induction on the number of iterations. $[\Gamma^k]$ is zero and trivially obeys all capacity constraints. Suppose $0 < i \leq k$ and $[\Gamma^i]$ is feasible; we show that $[\Gamma^{i-1}]$ is feasible. Note that $[\Gamma^{i-1}] = [\Gamma^i] + [\Delta^{i-1}]$. Consider the capacity constraint of edge $yz$ at time $\theta$. If $[\Delta^{i-1}]_{yz}(\theta) = 0$, then the induction hypothesis implies that this constraint is not violated. If $[\Delta^{i-1}]_{yz}(\theta) \neq 0$, then Lemma 6.3.3 implies that $\theta \geq p^i(y)$, and consequently Lemmas 6.3.2 and 6.3.4 imply that $[\Gamma^i]_{yz}(\theta) = f^i_{yz}$. Since $[\Delta^{i-1}]$ is a feasible dynamic flow in $\mathcal{N}^{i-1}_{f^i}$, the capacity constraint is not violated.

Finally, we consider flow conservation. These constraints are trivial except at the source nodes: the chain decomposition $\Gamma^0$ includes chains flows terminating at sources. However, the validity of the capacity constraints and the assumption that no source has incoming capacity in $\mathcal{N}$ guarantee that no source ever has net incoming flow. ■

**Theorem 6.3.6** $[\Gamma^0]$ has time horizon $T$.

**Proof:** First we prove that $[\Gamma^0]$ is a finite-horizon flow, and then we prove the theorem.

**(1)** $[\Gamma^0]$ *has a finite time horizon.*

For each edge $yz$, let $T_{yz}$ be the longest $(\psi, y)$-length of a chain flow in $\Gamma^0$ touching $yz$ (or zero if no chain flow in $\Gamma^0$ touches $yz$); clearly this value is finite. Because the final network $\mathcal{N}^0$ has no negative-cost cycles, Lemmas 6.3.1 and 3.1.4 imply that the final static flow $f^0$ is zero, so that $[\Gamma^0]_{yz}(\theta) = 0$ for any time $\theta \geq T_{yz}$. Dynamic flow $[\Gamma^0]$ is finished after $\max_{yz}\{T_{yz}\}$ time steps.

**(2)** $[\Gamma^0]$ *has time horizon* $T$.

Consider any sink $s_i \in S^-$; recall that artificial edge $s_i\psi$ has transit time $-(T+1)$. Let $ys_i$ be an edge in the input network $\mathcal{N}$. Every chain flow using $ys_i$ (forward) ends with edge $s_i\psi$, is part of a minimum-cost circulation, and so must reach sink

$s_i$ by time $T+1$; every chain flow using $s_i y$ ($y s_i$ backward) starts with edge $\psi s_i$ and so reaches sink $s_i$ at time $T+1$. Thus, $T_{s_i y} \leq T+1$ — that is, there is no flow entering any sink in the network after time $T$.

Assume for a contradiction that there is flow left somewhere in the network after time $T$. All edges with positive capacity have non-negative length, so that this flow cannot reach any sink by time $T$ and hence must stay in the network forever. This contradicts the fact that $[\Gamma^0]$ has a finite time horizon. ■

**Theorem 6.3.7** $[\Gamma^0]$ is a lexicographically maximum dynamic flow.

**Proof:** Given any index $i : 0 \leq i \leq k$, we prove that the amount of dynamic flow leaving the set of terminals $S_i = \{s_0, \ldots, s_{i-1}\}$ is maximum. Our proof relies on the infinite time-expanded network $\mathcal{N}(*)$. We construct a cut $C_i$ in this network that separates the source set $\{s_j(0) : s_j \in S_i^+\} \cup \{s_j(T) : s_j \in S_i^-\}$ from the sink set $\{s_j(0) : s_j \in S^+ \setminus S_i\} \cup \{s_j(T) : s_j \in S^- \setminus S_i\}$. We then show that $[\Gamma^0]$ saturates this cut and so sends as much flow as possible from $S_i$ to $S \setminus S_i$ within time horizon $T$.

The cut is defined as $C_i = \{y(\theta) : \theta \geq p^i(y)\} \cup \{s_j(T) : s_j \in S_i^-\}$. It is easy to see that $C_i$ contains the source set, because the artificial edges in network $\mathcal{N}^i$ guarantee that $p^i(s_j) = 0$ for all sources $s_j$ in $S_i^+$. It is not much harder to see that $C_i$ contains no element of the sink set by considering each iteration $j \geq i$. If terminal $s_j$ is in $S^-$, then iteration $j$ adds infinite-capacity artificial edge $s_j \psi$ with transit time $-(T+1)$ so that the subsequent minimum-cost circulation establishes $p^j(s_j) \geq T+1$; otherwise $s_j \in S^+$ and iteration $j$ saturates every residual $(\psi, s_j)$-path so that $p^j(s_j) = \infty$. These facts together with Lemma 6.3.2 imply that $p^i(s_j) > T$ for all terminals $s_j$ in $S \setminus S_i$, and so no terminal in the sink set $\{s_j(0) : s_j \in S^+ \setminus S_i\} \cup \{s_j(T) : s_j \in S^- \setminus S_i\}$ is contained in $C_i$.

Next we show that $[\Gamma^0]$ saturates cut $C_i$. Consider any edge $y(\theta)z(\theta')$ that crosses $C_i$; then $y(\theta) \in C_i$ and $z(\theta') \notin C_i$. (The other direction is considered as opposite edge $z(\theta')y(\theta)$.) There are three cases: the first applies when $\theta < p^i(y)$, otherwise we distinguish non-holdover edges from holdover edges. (1) If $\theta < p^i(y)$ then $y \in S_i^-$. Recall our assumption that each sink in $S^-$ has no outgoing capacity in the input network $\mathcal{N}$; this implies that the capacity of edge $yz$ is zero. (2) Consider any holdover edge $y(\theta)y(\theta')$ that crosses $C_i$. By the definition of $C_i$, this must be a reverse holdover edge with $\theta' = \theta - 1$. The capacity of a reverse holdover edge is zero. (3) Suppose $\theta \geq p^i(y)$ and $y(\theta)z(\theta')$ is not a holdover edge. Then $\theta' = \theta + \tau_{yz}$ and $y(\theta)z(\theta + \tau_{yz})$ crosses $C_i$ for every $\theta : p^i(y) \leq \theta < p^i(z) - \tau_{yz}$. Thus, the total capacity of cut $C_i$ is

$$\sum_{yz \in E^+ \setminus \psi} \max\{0, p^i(z) - \tau_{yz} - p^i(y)\} u_{yz}.$$

By Lemma 3.1.2 applied to edges $yz$ and $zy$, the above formula reduces to

$$\sum_{yz \in E^+ \setminus \psi} (p^i(z) - p^i(y)) f^i_{yz} - \sum_{yz \in E^+ \setminus \psi} \tau_{yz} f^i_{yz}.$$

Flow conservation implies that the first sum cancels itself out except at terminal nodes. Notice that for any source $s_j$ in $S^+$, if $f^i_{s_j z} > 0$ then $p^i(s_j) = 0$; and for any sink $s_j$ in $S^-$, if $f^i_{y s_j} > 0$ then $p^i(s_j) = T + 1$. Thus, we can simplify the remaining terms by considering all the artificial edges of network $\mathcal{N}^i$ and deriving the capacity of cut $C_i$ to be $-\sum_{yz \in E^+} \tau_{yz} f^i_{yz}$.

The value of $[\Gamma^0]$ across $C_i$ is equal to the net flow out of $S_i$, derived as follows: Observe that $f^i$ is the sum of all chain flows into $S \setminus S_i$. Therefore, Corollary 4.3.2 implies that the total cost of $f^i$ is equal to the dynamic flow value out of $S \setminus S_i$. By dynamic flow conservation, this value is exactly opposite to the net flow out of $S_i$. Thus, the capacity of cut $C_i$ is equal to the value of $[\Gamma^0]$ across it. ■

Note that the maximum dynamic flow problem is just the lexicographically maximum dynamic flow problem in a network with one source $s_0$ and one sink $s_1$. Theorem 4.2.3 is essentially a two-terminal special case of Theorem 6.3.7 that uses a standard chain-decomposable flow $[\Gamma]^T$ rather than a chain-decomposable flow $[\Gamma^0]$.

# Chapter 7

# The Dynamic Transshipment Problem

In the *dynamic transshipment problem*, we are given a dynamic network $\mathcal{N}$ with time horizon $T$ and supply vector $v$. We seek a feasible dynamic flow with time horizon $T$ that satisfies supply $v$, if such a flow exists. We denote an instance of the dynamic transshipment problem by the triplet $(\mathcal{N}, v, T)$.

In this chapter, we describe a polynomial algorithm to solve the integral dynamic transshipment problem. Our algorithm takes as input an integral instance of the dynamic transshipment problem, and for output finds an integral solution, if one exists. (An integral solution exists if and only if there is a fractional solution.) Our algorithm reduces a dynamic transshipment problem to an equivalent lex max dynamic flow problem in a slightly modified network. We described a polynomial lex max dynamic flow algorithm in Chapter 6.

Before discussing the main algorithm, we describe how to determine the feasibility of a dynamic transshipment problem in polynomial time. This test is the core subroutine in our main dynamic transshipment algorithm.

## 7.1 Dynamic Transshipment Feasibility

Ford and Fulkerson [16] proved perhaps the best-known result in network flow theory: the maximum value of an $(s, t)$-flow is equal to the minimum capacity of an $(s, t)$-cut in any network $\mathcal{N}$. This famous theorem implies a multi-terminal corollary: there exists a feasible flow satisfying supply vector $v$ if and only if for every subset of terminals $A \subseteq S$ the maximum flow jointly from $A$ to $S \setminus A$ has value no less than $v(A)$.

Klinz [33] derived similar results for dynamic network flows. Consider a dynamic network $\mathcal{N}$ with time horizon $T$. Recall the maximum value function $o(A)$ defined in Chapter 6 for terminal subsets $A \subseteq S$. For a dynamic transshipment problem $(\mathcal{N}, v, T)$ to be feasible, clearly we must have that $v(A) \leq o(A)$ for every terminal subset $A \subseteq S$. Klinz observed that this condition is equivalent to Ford and Fulkerson's cut condition of feasibility applied to time-expanded network $\mathcal{N}(T)$. Thus, this condition is not only necessary, but also sufficient.

It is a well-known result of Ford and Fulkerson [16] that single-commodity static network flow problems can always be solved by integral flows when the input data are integral. This fact applies equally to the dynamic transshipment problem, again for the reason that it is equivalent to a static network flow problem in $\mathcal{N}(T)$.

**Theorem 7.1.1** (Klinz [33]) The dynamic transshipment problem $(\mathcal{N}, v, T)$ is feasible if and only if $v(A) \leq o(A)$ for every subset $A \subseteq S$. Furthermore, if the problem is feasible and the input data are integral, then there is an integral solution.

**Violated Sets.** Given a dynamic transshipment problem, we call a subset of terminals $A \subseteq S$ *violated* if $v(A) > o(A)$. Theorem 7.1.1 implies that we can test the feasibility of a dynamic transshipment problem simply by searching for a violated set of terminals. Our dynamic transshipment algorithm requires a subroutine that not only determines feasibility of a dynamic transshipment problem but also finds a violated set if the problem is infeasible. A simple exhaustive search, testing all possible subsets $A \subseteq S$, yields the following result:

**Lemma 7.1.2** For any dynamic transshipment problem $(\mathcal{N}, v, T)$, a violated set can be found (or feasibility proven) in $O(2^k \,\mathrm{MCF}\,)$ time.

**Proof:** There are $2^k$ subsets $A \subseteq S$. For any particular subset of terminals $A$, computing $o(A)$ reduces directly to a single-source single-sink maximum dynamic flow problem, and so Corollary 4.2.2 and Theorem 4.2.3 imply that we can determine if $v(A) > o(A)$ in $O(\,\mathrm{MCF}\,)$ time. ■

In the remainder of this section, we derive theoretically efficient alternatives to this brute-force algorithm. They are geometric algorithms that work with the *feasibility polytope* induced by the given dynamic transshipment problem; we discuss this polytope in the next subsection. Unfortunately, these algorithms are much more complicated than the simple approach just described. In networks with only a few terminals, exhaustive search should be preferable to currently-known polynomial techniques for feasibility testing.

### 7.1.1 The Feasibility Polytope

Consider a dynamic network $\mathcal{N}$ with time horizon $T$. Recall from Chapter 6 that $\mathcal{P}$ denotes the set of feasible supply vectors for this network and time horizon. Testing the feasibility of a supply vector $v$ is equivalent to determining if $v \in \mathcal{P}$. In this subsection we prove that $\mathcal{P}$ is a *polytope*, which is a bounded set of the form $\{v \in \mathbb{R}^S : \mathcal{A}v \leq b\}$.

First, we consider the vector space spanned by $\mathcal{P}$. (The origin is contained in $\mathcal{P}$, so this is equivalent to the affine hull of $\mathcal{P}$.) Notice that $\mathcal{P}$ must be contained in the $(k-1)$-dimensional subspace $\{v \in \mathbb{R}^S : v(S) = 0\}$, since we require that the total supply of all terminals be zero. The subspace actually spanned by $\mathcal{P}$ can be determined by a generalization of this observation. Suppose the terminal set can be partitioned $S = A \cup B$ so that no flow can possibly go from $A$ to $B$ or from $B$

to $A$ through network $\mathcal{N}$ within time horizon $T$; then any supply vector $v$ cannot be feasible unless $v(A) = v(B) = 0$.

We seek such a partition of $S$ into as many subsets as possible. We can construct the partition $S = S^1 \cup \ldots \cup S^h$ with $O(k^2)$ shortest path computations in network $\mathcal{N}$. Our construction relies on a bipartite graph using the sources $S^+$ of $\mathcal{N}$ as one node set and the sinks $S^-$ of $\mathcal{N}$ as the other node set. For each source-sink pair $s_i \in S^+, s_j \in S^-$, connect $s_i$ and $s_j$ by an edge if and only if the shortest $(s_i, s_j)$-path in $\mathcal{N}$ is no longer than $T$. The subsets $S^1, \ldots, S^h$ are the connected components of this bipartite graph. We say a subset $S^i$ is *trivial* if it consists of one terminal; the supply of any terminal in a trivial subset must be zero.

Let $Z$ denote the subspace $\{v \in \mathbb{R}^S : v(S^i) = 0 \ \ \forall i = 1, \ldots, h\}$. We prove in Lemma 7.1.3 that $\mathcal{P}$ spans this space. We also prove that $\mathcal{P}$ is contained within a ball centered close to the origin and that $\mathcal{P}$ contains a smaller ball in space $Z$. Along with the convexity of $\mathcal{P}$, these properties are enough for us to determine efficiently if $v \in \mathcal{P}$ for any $v$ in $\mathbb{R}^S$. The rest of this subsection depends on the following definitions. Let

$F$ be a spanning forest of the bipartite graph just described;

$\mathcal{I}^F$ be the $k \times (k-h)$ node-edge incidence matrix of $F$: $\mathcal{I}^F_{ij} = 1$ (or $-1$) if edge $j$ leaves (or enters) node $i$ in $F$, and $\mathcal{I}^F_{ij} = 0$ otherwise; and

$\hat{x}$ be $1/2k$ times the sum of the columns of $\mathcal{I}^F$.

For any norm $p$, $\bar{x} \in Z$, and $r > 0$, let $\mathcal{B}^Z_p(\bar{x}, r)$ denote the closed ball $\{x \in Z : \|x - \bar{x}\|_p \leq r\}$.

**Lemma 7.1.3** $\mathcal{P}$ is a full-dimensional polyhedron in subspace $Z$.

**Proof:** It is clear that $Z$ is a $(k-h)$-dimensional subspace and that $\mathcal{P}$ is contained in this space. We claim that $\mathcal{P}$ is a full-dimensional polyhedron in this subspace. The columns of $\mathcal{I}^F$ are linearly independent because $F$ is a forest; notice also that any column of $\mathcal{I}^F$ is a vector in $\mathcal{P}$; this implies that $\mathcal{P}$ is full-dimensional in $Z$.

Theorem 7.1.1 implies that $\mathcal{P}$ is a polyhedron, which is a (not necessarily bounded) set of the form $\{v \in \mathbb{R}^S : \mathcal{A}v \leq b\}$. The rows of $\mathcal{A}$ are the characteristic vectors of the terminal subsets $A \subseteq S$, and the entries of $b$ are the values $o(A)$ for each of these subsets. There is one additional constraint $v(S) \geq 0$. ■

**Lemma 7.1.4** $\mathcal{P}$ is a polytope. It is contained within $\mathcal{B}_2^Z(\hat{x}, \sqrt{k}nU(T+2))$.

**Proof:** No terminal can send or receive more than $nU$ units of flow per time step, from time zero till time $T$; this implies that $\mathcal{P} \subseteq \mathcal{B}_\infty^Z(0, nU(T+1))$. Because there are only $k-h$ edges in $F$, we have $\|\hat{x}\|_\infty < 1/2$, which implies $\mathcal{P} \subseteq \mathcal{B}_\infty^Z(\hat{x}, nU(T+2))$. The lemma then follows from the fact that $\|x\|_2 \leq \sqrt{k}\|x\|_\infty$ for all $x \in \mathbb{R}^k$. ■

**Lemma 7.1.5** $\mathcal{P}$ contains $\mathcal{B}_2^Z(\hat{x}, 1/2k)$.

**Proof:** Because $\|x\|_\infty \leq \|x\|_2$ for all $x$, we can prove the lemma using infinity-norm boxes. If $s_j$ is a source in some non-trivial subset $S^i$, then there is at least one edge of $F$ leaving $s_j$, so that $\hat{x}_{s_j} \geq 1/2k$. Similarly, $\hat{x}_{s_j} \leq -1/2k$ for any sink $s_j$ in a non-trivial subset $S^i$. Thus, if $x \in \mathcal{B}_\infty^Z(\hat{x}, 1/2k)$ then $x_{s_j} \geq 0$ for any source $s_j$ and $x_{s_j} \leq 0$ for any sink $s_j$. Because there are only $k-h$ edges in $F$, we have $\hat{x}(S^+) < 1/2$. Thus, if $x \in \mathcal{B}_\infty^Z(\hat{x}, 1/2k)$ then $x(S^+) < 1$.

The lemma finally follows from the fact that if $x_{s_j} \geq 0$ for every source $s_j$, $x_{s_j} \leq 0$ for every sink $s_j$, $x(S^+) \leq 1$, and $x \in Z$, then $x \in \mathcal{P}$. Such an $x$ can be expressed as a convex combination of the columns of $\mathcal{I}^F$ along with the zero vector. ■

### 7.1.2 A Polynomial Algorithm for Feasibility Testing

Given a dynamic transshipment problem $(\mathcal{N}, v, T)$, we want to determine if $v$ lies in the feasibility polytope $\mathcal{P}$ or not. Grötschel *et al* [22] proved that there is a polynomial algorithm to test membership in a convex set if and only if there is a polynomial algorithm to optimize a linear objective function over that set. We proved in Theorems 6.1.2 and 6.2.1 that there is a polynomial algorithm to optimize over the convex set $\mathcal{P}$; thus, we arrive at the following result:

**Theorem 7.1.6** For any dynamic transshipment problem $(\mathcal{N}, v, T)$, a violated set can be found (or feasibility proven) in polynomial time.

In most applications, testing membership in a convex set is considered an easier problem than optimizing over that set. The usual application of [22] is to construct an optimization algorithm, given an oracle to test membership. We are in the opposite situation. Our lex max dynamic flow algorithm gives us an efficient combinatorial algorithm to optimize over $\mathcal{P}$, but we do not have an efficient combinatorial algorithm to test membership in $\mathcal{P}$. Nevertheless, Theorem 7.1.6 implies that a (not necessarily combinatorial) efficient algorithm does exist. We could use the ellipsoid method as described in [22] to derive a polynomial algorithm for testing feasibility; however, we obtain a more efficient algorithm by relying on an algorithm of Vaidya [52] instead.

Vaidya states his result as an efficient optimization algorithm given an oracle to test membership. Because we are building a membership algorithm from an optimization oracle, we apply Vaidya's algorithm to the *polar* of $\mathcal{P}$, defined as $\mathcal{P}^* = \{w \in \mathbb{R}^S : x^T w \leq 1 \;\; \forall x \in \mathcal{P}\}$. Polars are characterized by the following well-known fact:

**Lemma 7.1.7** Suppose $\mathcal{P}$ is a closed convex set containing the origin. Then $x \in \mathcal{P}$ if and only if $\forall w \in \mathcal{P}^* : x^T w \leq 1$.

Lemma 7.1.7 implies that testing membership in $\mathcal{P}$ reduces to a linear optimization problem over the convex set $\mathcal{P}^*$. The value $\max\{v^T w : w \in \mathcal{P}^*\}$ is no more than one if and only if the given dynamic transshipment problem is feasible. In order to solve this optimization problem, Vaidya's algorithm needs to know the vector space spanned by $\mathcal{P}^*$ and the radii of two balls: one contained within $\mathcal{P}^*$ and another centered at the origin that contains $\mathcal{P}^*$; it also needs an oracle to test membership in $\mathcal{P}^*$. The membership oracle must not only determine membership in the set but also provide a separating hyperplane when given any point not in the set.

Because $\mathcal{P}$ is a full-dimensional polytope in $(k-h)$-dimensional subspace $Z$, the polar $\mathcal{P}^*$ contains $h$-dimensional subspace $Z^\perp = \{w \in \mathbb{R}^S : w^T x = 0 \;\; \forall x \in Z\}$ and $\mathcal{P}^* \cap Z$ is a full-dimensional convex set containing no non-trivial subspace. Unfortunately, $\mathcal{P}^* \cap Z$ is not bounded. In order to obtain a bounded convex set, we consider an affine variant of the polar restricted to subspace $Z$. For any $\bar{x}$ in $Z$, let

$$\mathcal{P}^*_{Z\bar{x}} = \{w \in Z : (x - \bar{x})^T w \le 1 \;\; \forall x \in \mathcal{P}\}.$$

We run Vaidya's algorithm on the set $\mathcal{P}^*_{Z\hat{x}}$. Our result depends on the following technical lemmas, which we prove at the end of this subsection:

**Lemma 7.1.8** Let $\bar{x} \in Z$. $x \in \mathcal{P}$ if and only if $x \in Z$ and $\forall w \in \mathcal{P}^*_{Z\bar{x}} : (x - \bar{x})^T w \le 1$.

**Lemma 7.1.9** $\mathcal{P}^*_{Z\hat{x}}$ contains $\mathcal{B}^Z_2(0, (\sqrt{k} nU(T+2))^{-1})$.

**Lemma 7.1.10** $\mathcal{P}^*_{Z\hat{x}}$ is contained within $\mathcal{B}^Z_2(0, 2k)$.

**Lemma 7.1.11** Suppose $w \in Z$. The lex max dynamic flow algorithm determines if $w \in \mathcal{P}^*_{Z\hat{x}}$ in $O(k\,\mathrm{MCF})$ time. If $w \notin \mathcal{P}^*_{Z\hat{x}}$, then it computes a vector $c$ in $\mathbb{R}^S$ such that $\max\{c^T y : y \in \mathcal{P}^*_{Z\hat{x}}\} \le c^T w$.

**Theorem 7.1.12** For any dynamic transshipment problem $(\mathcal{N}, v, T)$, a violated set can be found (or feasibility proven) in $O(k^2\,\mathrm{MCF}\log(nUT))$ time.

**Proof:** We assume $v \in Z$ (it is easy to check this and find a violated set if $v \notin Z$). We use Vaidya's algorithm to solve $\max\{(v - \hat{x})^T w : w \in \mathcal{P}^*_{Z\hat{x}}\}$. The algorithm works on any full-dimensional convex set contained in a ball of radius $2^L$ centered at the origin and containing a ball of radius $2^{-L}$. The algorithm takes $O(kL)$ iterations given a $k$-dimensional set. Each iteration inverts a $k \times k$ matrix and calls the membership oracle at most once.

Lemmas 7.1.9 and 7.1.10 imply that $L = O(\log(nUT))$. By Lemma 7.1.11, our lex max dynamic flow algorithm provides a membership oracle; notice that its $O(k\,\mathrm{MCF})$ running time dominates the time required to invert a $k \times k$ matrix. The total running time of the algorithm is $O(k^2\,\mathrm{MCF}\log(nUT))$.

If $\max\{(v-\hat{x})^T w : w \in \mathcal{P}^*_{Z\hat{x}}\}$ is at most one, then Lemma 7.1.8 implies that $v \in \mathcal{P}$ and the dynamic transshipment problem is feasible. Otherwise, Vaidya's algorithm returns $w$ in $\mathcal{P}^*_{Z\hat{x}}$ such that $(v-\hat{x})^T w > 1$. We claim that $S(\alpha) = \{s_i \in S : w_{s_i} \geq \alpha\}$ is a violated set for some $\alpha$.

Assume for notational simplicity that $w_{s_0} \geq w_{s_1} \geq \ldots \geq w_{s_{k-1}}$; then $S(w_{s_i}) = S_{i+1}$ as we defined $S_{i+1}$ in Chapter 6. The inner product of $w$ with $v$ is

$$w^T v \quad = \quad w_{s_{k-1}} v(S_k) + \sum_{i=1}^{k-1} (w_{s_{i-1}} - w_{s_i}) v(S_i).$$

Now consider maximizing $w^T x$ over all $x$ in $\mathcal{P}$. By Theorem 6.1.2, this is a lex max dynamic flow problem and the maximizer $x^*$ in $\mathcal{P}$ has objective function value

$$w^T x^* \quad = \quad w_{s_{k-1}} o(S_k) + \sum_{i=1}^{k-1} (w_{s_{i-1}} - w_{s_i}) o(S_i).$$

Because $x^* \in \mathcal{P}$ and $w \in \mathcal{P}^*_{Z\hat{x}}$, we have $w^T x^* \leq 1 + \hat{x}^T w$; we also assume $w^T v > 1 + \hat{x}^T w$. Therefore, the formulas above imply that $v(S_i) > o(S_i)$ for some $i$. Finally, notice that the time required to run Vaidya's algorithm dominates both the time to compute $\hat{x}$ and the time it takes to find a violated set. $\blacksquare$

We conclude this subsection by proving some technical lemmas.

**Proof of Lemma 7.1.8:** Suppose $x \in \mathcal{P}$. Then Lemma 7.1.3 implies that $x \in Z$ and the definition of $\mathcal{P}^*_{Z\bar{x}}$ implies that $(x-\bar{x})^T w \leq 1$ for all $w \in \mathcal{P}^*_{Z\bar{x}}$.

To prove the converse, we consider Lemma 7.1.7 applied to the polytope $\mathcal{P}_{\bar{x}} = \{x - \bar{x} : x \in \mathcal{P}\}$ with traditional polar $(\mathcal{P}_{\bar{x}})^*$. The lemma is trivial if $x \notin Z$; otherwise, if $x \in Z \setminus \mathcal{P}$ then $x - \bar{x} \notin \mathcal{P}_{\bar{x}}$ and there is a $w$ in $(\mathcal{P}_{\bar{x}})^* \cap Z$ such that $(x-\bar{x})^T w > 1$. The lemma follows from the observation that $(\mathcal{P}_{\bar{x}})^* \cap Z = \mathcal{P}^*_{Z\bar{x}}$. $\blacksquare$

The following two lemmas help prove Lemmas 7.1.9 and 7.1.10:

**Lemma 7.1.13** $\mathcal{P} \subseteq \mathcal{B}_2^Z(\bar{x}, R) \;\Rightarrow\; \mathcal{B}_2^Z(0, 1/R) \subseteq \mathcal{P}^*_{Z\bar{x}}$ for any $\bar{x} \in Z$ and $R > 0$.

**Proof:** Suppose $\mathcal{P} \subseteq \mathcal{B}_2^Z(\bar{x}, R)$. Let $w \in \mathcal{B}_2^Z(0, 1/R)$ and $x \in \mathcal{P}$. By the well-known Schwarz inequality, $|(x-\bar{x})^T w| \leq \|x-\bar{x}\|_2 \|w\|_2$. Our assumptions imply that this value is no more than one, so that $w \in \mathcal{P}^*_{Z\bar{x}}$. ■

**Lemma 7.1.14** $\mathcal{B}_2^Z(\bar{x}, r) \subseteq \mathcal{P} \;\Rightarrow\; \mathcal{P}^*_{Z\bar{x}} \subseteq \mathcal{B}_2^Z(0, 1/r)$ for any $\bar{x} \in Z$ and $r > 0$.

**Proof:** Suppose $\mathcal{B}_2^Z(\bar{x}, r) \subseteq \mathcal{P}$. Let $w \in \mathcal{P}^*_{Z\bar{x}}$ and suppose $\|w\|_2 > 0$ (otherwise $w \in \mathcal{B}_2^Z(0, 1/r)$ is trivial). Consider $x = \bar{x} + rw/\|w\|_2$. Because $\|x - \bar{x}\|_2 = r$ it follows that $x \in \mathcal{P}$ and so $(x-\bar{x})^T w \leq 1$. Because $x - \bar{x}$ is a positive scalar multiple of $w$, we have $(x-\bar{x})^T w = \|x-\bar{x}\|_2 \|w\|_2 = r\|w\|_2$. Thus $\|w\|_2 \leq 1/r$. ■

**Proof of Lemmas 7.1.9 and 7.1.10:** Lemmas 7.1.4 and 7.1.13 imply that $\mathcal{P}^*_{Z\hat{x}}$ contains $\mathcal{B}_2^Z(0, (\sqrt{k}nU(T+2))^{-1})$. Lemmas 7.1.5 and 7.1.14 imply that $\mathcal{P}^*_{Z\hat{x}}$ is contained within $\mathcal{B}_2^Z(0, 2k)$. ■

**Proof of Lemma 7.1.11:** By Theorem 6.1.2, we can compute $\max\{w^T x : x \in \mathcal{P}\}$ via one lex max dynamic flow computation. Lemma 7.1.8 implies that $w \in \mathcal{P}^*_{Z\hat{x}}$ if and only if the maximum value is at most $1 + \hat{x}^T w$. If $w \notin \mathcal{P}^*_{Z\hat{x}}$ then the lex max dynamic flow computation returns supply vector $x^*$ in $\mathcal{P}$ such that $(x^* - \hat{x})^T w > 1$. By the definition of $\mathcal{P}^*_{Z\hat{x}}$, this means that $\max\{(x^*-\hat{x})^T y : y \in \mathcal{P}^*_{Z\hat{x}}\} < (x^*-\hat{x})^T w$. Theorem 6.2.1 implies that this procedure takes $O(k\,\mathrm{MCF})$ time. ■

### 7.1.3 A Strongly Polynomial Algorithm

Our strongly polynomial algorithm is a simple corollary of Theorem 7.1.1 and the following two results of Megiddo [37] and Grötschel *et al* [22]:

**Theorem 7.1.15** (Megiddo [37]) $o(A) + o(B) \geq o(A \cap B) + o(A \cup B)$ for every pair of subsets $A, B \subseteq S$ in any dynamic network $\mathcal{N}$ with time horizon $T$. That is, function $o$ is submodular.

**Theorem 7.1.16** (Grötschel *et al* [22]) Suppose $f : 2^S \to \mathbb{R}$ is a submodular function. A subset $W \subset S$ minimizing $f(W)$ can be found in strongly polynomial time.

**Corollary 7.1.17** For any dynamic transshipment problem $(\mathcal{N}, v, T)$, a violated set can be found (or feasibility proven) in strongly polynomial time.

**Proof:** Consider the submodular function $b(A) = o(A) - v(A)$. A violated set is a set $A$ such that $b(A)$ is negative. Therefore, a violated set can be found (or feasibility proven) by minimizing the submodular function $b$. ■

**Remark on Extended Polymatroids.** Any submodular function $f : 2^S \to \mathbb{R}$ induces a polyhedron $EP_f = \{v \in \mathbb{R}^S : v(A) \leq f(A) \;\; \forall A \subseteq S\}$. This polyhedron is called an *extended polymatroid* and it has a special structure ideal for optimizing linear objective functions, as discovered by Edmonds [14].

Given any vector $c$ in $\mathbb{R}^S$, order the elements of $S$ so that $c_{s_0} \geq c_{s_1} \geq \ldots \geq c_{s_{k-1}}$. Edmonds proved that a vector $v^*$ maximizing $c^T v$ over all vectors $v$ in $EP_f$ can be found by the *greedy algorithm*:

$$\begin{aligned}
v^*_{s_0} &= \max\{v_{s_0} : v \in EP_f\} \\
v^*_{s_1} &= \max\{v_{s_1} : v \in EP_f \text{ and } v_{s_0} = v^*_{s_0}\} \\
&\ldots \\
v^*_{s_{k-1}} &= \max\{v_{s_{k-1}} : v \in EP_f \text{ and } v_{s_0} = v^*_{s_0}, \ldots, v_{s_{k-2}} = v^*_{s_{k-2}}\}.
\end{aligned}$$

For any dynamic transshipment problem $(\mathcal{N}, v, T)$, Theorems 7.1.1 and 7.1.15 imply that the feasibility polytope $\mathcal{P}$ is a facet of the extended polymatroid $EP_o$, more specifically $\mathcal{P} = \{v \in \mathbb{R}^S : v(S) = o(S) \text{ and } v(A) \leq o(A) \;\; \forall A \subseteq S\}$. Our lex max dynamic flow algorithm is a greedy algorithm. Theorem 6.1.2, which proves that the lex max dynamic flow algorithm optimizes linear objective functions over $\mathcal{P}$, also follows from the fact that $\mathcal{P}$ is a facet of an extended polymatroid.

## 7.2 Dynamic Transshipment Algorithm

In this section we describe a polynomial algorithm for the dynamic transshipment problem; in the next section we prove the running time and correctness of our algorithm. The algorithm relies on feasibility testing as a black box; we use FEAS to denote the time required to test feasibility of a dynamic transshipment problem.

### 7.2.1 Overview of Algorithm

Our algorithm reduces any dynamic transshipment problem $(\mathcal{N}, v, T)$ to an equivalent lex max dynamic flow problem on a slightly more complicated network $\mathcal{N}'$. We described a polynomial lex max dynamic flow algorithm in Chapter 6.

**Tight Sets.** Given a dynamic transshipment problem, we call a subset of terminals $A \subseteq S$ *tight* if $v(A) = o(A)$. Tight sets characterize a very special type of dynamic transshipment problem that reduces directly to lex max dynamic flow:

**Theorem 7.2.1** Suppose $\mathcal{C}$ is a chain of nested tight subsets of terminals in dynamic transshipment problem $(\mathcal{N}, v, T)$, and $|\mathcal{C}| = k + 1$. Then the dynamic transshipment problem can be solved by a single lex max dynamic flow in $\mathcal{N}$.

**Proof:** If $|\mathcal{C}| = k + 1$, then we can relabel the set of terminals $S$ so that $S_i = \{s_0, \ldots, s_{i-1}\}$ is a tight set in $\mathcal{C}$ for all $i = 0, \ldots, k$. Lemma 6.1.1 and the definition of tight sets imply that any solution $f$ to the lex max dynamic flow problem $(\mathcal{N}, \mathcal{C}, T)$ satisfies $|f(S_i)|_T = v(S_i)$ for all $i = 0, \ldots, k$ and so yields a solution to the dynamic transshipment problem. ■

Most dynamic transshipment problems do not satisfy the requirements of Theorem 7.2.1 to reduce directly to lex max dynamic flow. Next, we describe how to modify any dynamic transshipment problem $(\mathcal{N}, v, T)$ so that it does satisfy the requirements of Theorem 7.2.1. Our algorithm attaches to the network a new set of terminals $S'$ and creates for them a new supply vector $v'$. Initially, $S'$ contains one

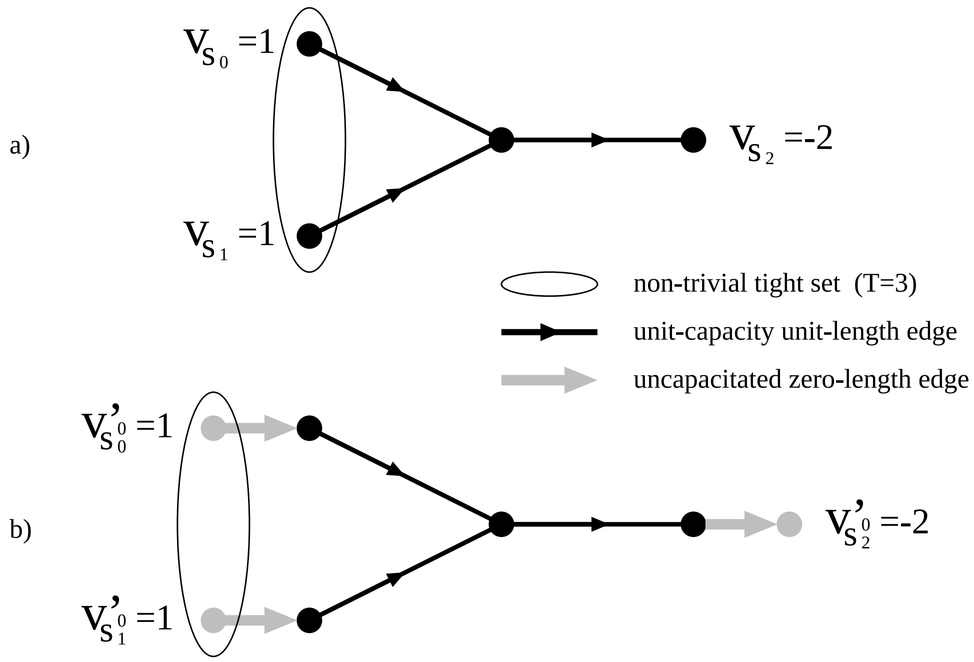


Figure 7.1: Transforming $(\mathcal{N}, v, T)$ to $(\mathcal{N}', v', T)$

source $s_i^0$ for each source $s_i$ in $S$, connected by edge $s_i^0 s_i$ with infinite capacity and zero transit time; likewise, $S'$ contains one sink $s_i^0$ for each sink $s_i$ in $S$, connected by edge $s_i s_i^0$ with infinite capacity and zero transit time. Define $v'$ in $\mathbb{R}^{S'}$ based on $v$ in the obvious manner: $v'_{s_i^0} = v_{s_i}$. Clearly $(\mathcal{N}', v', T)$ is equivalent to $(\mathcal{N}, v, T)$.

**Example:** Figure 7.1(a) depicts dynamic transshipment problem $(\mathcal{N}, v, T)$ with time horizon $T = 3$. There is one non-trivial tight set. (Trivial tight sets are $\emptyset$ and $S$.) Figure 7.1(b) shows the corresponding problem $(\mathcal{N}', v', T)$.

The algorithm maintains a chain $\mathcal{C}$ whose elements are tight subsets of $S'$ ordered by inclusion. The goal of the algorithm is to extend $\mathcal{C}$ to a complete chain of size $|S'| + 1$. Theorem 7.2.1 implies that a complete chain $\mathcal{C}$ reduces the associated dynamic transshipment problem to a single lex max dynamic flow problem on the same network.

The body of the algorithm is a loop. Each iteration of the loop adds new terminals to $S'$. Each new terminal $s_i^j$ is attached to a terminal $s_i$ of the original network $\mathcal{N}$ by an edge as described above, but using finite capacities and non-negative transit times. These capacities and transit times restrict the ability of new terminals to send or receive flow; the algorithm assigns a supply to each new terminal $s_i^j$ based on these restrictions and adjusts the supply of the corresponding $s_i^0$ so that for any original terminal $s_i$ in $S$, the total supply of new terminals in $S'$ associated with $s_i$ is $v_{s_i}$. This maintains the invariant that any solution to $(\mathcal{N}', v', T)$ yields a solution to $(\mathcal{N}, v, T)$.

We measure the progress of the algorithm by the formula $|\mathcal{C}| - |S'|$. Initially, $\mathcal{C} = \{\emptyset, S'\}$, and so $|\mathcal{C}| - |S'| = 2 - k$. By Theorem 7.2.1, we are done when $|\mathcal{C}| - |S'| = 1$, and so we need to increase the value of this expression. In the next subsection, we describe how one iteration of the algorithm increases $|\mathcal{C}| - |S'|$ by one and maintains the feasibility of $(\mathcal{N}', v', T)$. Thus, the given dynamic transshipment problem is reduced to an equivalent lex max dynamic flow problem after $k - 1$ iterations.

**Network Inversion.** In a straightforward implementation of our algorithm, each iteration could add either new sources or new sinks to the network. The treatment of sources and sinks is entirely symmetric, however, and so describing both cases with proofs would be somewhat tedious. To simplify our presentation, we make this symmetry precise by using *network inversion*, and we describe only the treatment of sources by the algorithm.

For any dynamic network $\mathcal{N}$, the inverted network $\mathcal{N}^{-1}$ consists of exactly the same nodes, edges, and terminals; but the capacities and transit times of edges are reversed: $u_{yz}^{-1} = u_{zy}$ and $\tau_{yz}^{-1} = \tau_{zy}$. For any family of subsets $\mathcal{C}$ in a universe $S$, we denote the inverted family $\mathcal{C}^{-1} = \{S \setminus A : A \in \mathcal{C}\}$. The symmetry of sources and sinks is captured by the following lemma of Minieka:

**Lemma 7.2.2** (Minieka [40]) Suppose flow $f$ solves lex max dynamic flow problem $(\mathcal{N}, \mathcal{C}, T)$. Then the inverted dynamic flow $f_{yz}^{-1}(\theta) = -f_{yz}(T-\theta)$ solves the inverted lex max dynamic flow problem $(\mathcal{N}^{-1}, \mathcal{C}^{-1}, T)$.

### 7.2.2 One Iteration of the Algorithm

An iteration begins with a modified dynamic transshipment problem $(\mathcal{N}', v', T)$. Network $\mathcal{N}'$ contains terminal set $S'$. In addition, each iteration starts with a chain $\mathcal{C}$ of tight subsets of $S'$ ordered by inclusion. The goal of an iteration is to increase $|\mathcal{C}| - |S'|$ by one. The first step toward this goal is to add more sources to $S'$. Done arbitrarily, these sources would take the algorithm farther from the goal; but new sources are connected to the network through edges with carefully computed capacities and transit times. By assigning supplies to new sources according to these capacities and transit times (and adjusting the supplies of other sources appropriately) the algorithm creates a modified but equivalent problem with enough new tight sets so that $|\mathcal{C}| - |S'|$ increases by one.

Let $Q \subset R$ be adjacent sets in $\mathcal{C}$ such that $|R \setminus Q| > 1$. (By adjacent we mean $\nexists A \in \mathcal{C} : \; Q \subset A \subset R$.) If no such $Q$ and $R$ exist, then $|\mathcal{C}| - |S'| = 1$ and we are done. The algorithm maintains the invariant that every terminal in $R \setminus Q$ is one of the first terminals $s_i^0$ in $S'$.

Let $s_i^0$ be a source in $R \setminus Q$. Lemma 7.2.2 implies that we lose no generality by assuming there is a source in $R \setminus Q$. (If there are no sources in $R \setminus Q$, then we can invert network $\mathcal{N}'$, negate supply vector $v'$, and invert chain $\mathcal{C}$ to obtain an equivalent problem with at least one source in $(S' \setminus Q) \setminus (S' \setminus R)$.)

We first check if the set $Q \cup \{s_i^0\}$ is tight; if so, then we can add it to chain $\mathcal{C}$ and the current iteration is done. If the set $Q \cup \{s_i^0\}$ is not tight, however, then we use the following steps to increase $|\mathcal{C}| - |S'|$ by one.

Source $s_i^0$ is adjacent to $s_i$, one of the original sources in $S$; suppose there are $j-1$ other sources in $S'$ adjacent to $s_i$. Add a new source $s_i^j$ to $S'$, connected

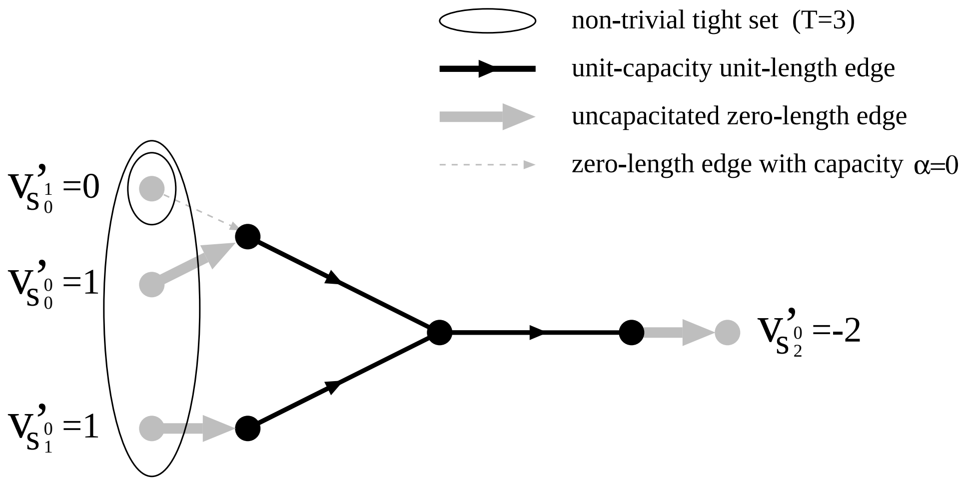


Figure 7.2: Connecting New Source $s_0^1$ with Parameterized Capacity

to $s_i$ by a parameterized $\alpha$-capacity zero-length edge. Let $Q' = Q \cup \{s_i^j\}$. Recall that $o(A)$ is the maximum dynamic flow out of a subset of terminals $A$. Define new supply $v^\alpha$: the supply of $s_i^j$ is $o(Q') - o(Q)$; the supply of $s_i^0$ is reduced by $o(Q') - o(Q)$; and other terminal supplies remain unchanged.

Consider the above parameterized dynamic transshipment problem $(\mathcal{N}^\alpha, v^\alpha, T)$. Notice that if $\alpha = 0$, then the maximum flow out of $s_i^j$ is zero: the problem is equivalent to $(\mathcal{N}', v', T)$ and is therefore feasible. At the opposite extreme, if $\alpha = \infty$ then $(\mathcal{N}^\alpha, v^\alpha, T)$ must be infeasible — if it were feasible then $Q \cup \{s_i^0\}$ must be a tight set, but we have already determined that $Q \cup \{s_i^0\}$ is not tight. Thus, we can binary search for an integer $\alpha^*$ such that $\alpha = \alpha^*$ yields a feasible problem but $\alpha = \alpha^* + 1$ yields an infeasible problem. An upper bound for this binary search is the total capacity out of the original source $s_i$ in $S$.

**Example:** Figure 7.2 shows the result of choosing source $s_0^0$ in the network of Figure 7.1(b) and completing the $\alpha$-binary search to determine the capacity of edge $s_0^1 s_0$. The critical value is $\alpha^* = 0$, which gives source $s_0^1$ a supply of zero. Note that choosing $\alpha = 1$ would give $s_0^1$ a supply of $o(\{s_0^1\}) - o(\emptyset) = 2$, which leaves $s_0^0$ with a supply of $-1$, which is infeasible (because $s_0^0$ has no incoming capacity).

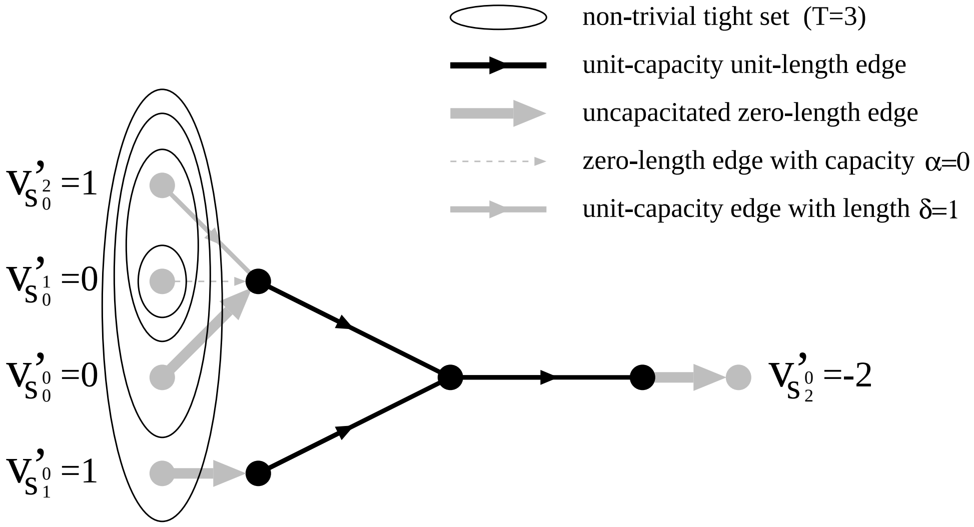


Figure 7.3: Connecting New Source $s_0^2$ with Parameterized Transit Time

Consider the dynamic transshipment problem $(\mathcal{N}^{\alpha^*}, v^{\alpha^*}, T)$. In this network, add another source $s_i^{j+1}$ to $S'$, connected to $s_i$ by a parameterized $\delta$-length unit-capacity edge. Let $Q'' = Q' \cup \{s_i^{j+1}\}$. Define new supply $\overline{v}^{\delta}$: the supply of $s_i^{j+1}$ is $o(Q'') - o(Q')$; the supply of $s_i^0$ is reduced by $o(Q'') - o(Q')$; and other terminal supplies remain unchanged.

Now we have a new parameterized dynamic transshipment problem $(\overline{\mathcal{N}}^{\delta}, \overline{v}^{\delta}, T)$. Notice that if $\delta = T+1$, then the maximum flow out of $s_i^{j+1}$ is zero: the problem is equivalent to the old capacity-parameterized problem $(\mathcal{N}^{\alpha^*}, v^{\alpha^*}, T)$ and is therefore feasible. On the other hand, if $\delta = 0$, then the problem is equivalent to the old capacity-parameterized problem $(\mathcal{N}^{\alpha^*+1}, v^{\alpha^*+1}, T)$ and is therefore infeasible. Thus, we can binary search for an integer $\delta^*$ such that $\delta = \delta^*$ yields a feasible problem but $\delta = \delta^* - 1$ yields an infeasible problem.

**Example:** Figure 7.3 shows the completion of a one-iteration run of the algorithm. (The rest of the run is illustrated in Figures 7.1 and 7.2.) The $\delta$-binary search determines the transit time of edge $s_0^2 s_0$. The critical value is $\delta^* = 1$, which gives source $s_0^2$ a supply of $o(\{s_0^1, s_0^2\}) - o(\{s_0^1\}) = 1$, which leaves $s_0^0$ with a supply of zero. Notice that the final network has a complete chain of nested tight subsets (including the trivial tight sets $\emptyset$ and $S'$) so that the resulting dynamic transshipment problem can be solved with a single lex max dynamic flow by Theorem 7.2.1.

So far, we have described how one iteration adds two new sources to dynamic transshipment problem $(\mathcal{N}', v', T)$ to obtain $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$, which is used in the next iteration. Chain $\mathcal{C}$ contains tight sets for the former problem and has not yet been modified. The progress of the algorithm depends on the following properties of the new dynamic transshipment problem $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$:

**Property 7.2.3** $Q'$ and $Q''$ are both tight sets.

**Property 7.2.4** If $A \in \mathcal{C}$ and $A \subseteq Q$, then $A$ is still a tight set.

**Property 7.2.5** If $A \in \mathcal{C}$ and $Q \subseteq A$, then $Q'' \cup A$ is a tight set.

**Property 7.2.6** There exists a tight set $W$ such that $Q'' \subset W \subset (Q'' \cup R)$.

The first three properties follow from definitions. In the next section, we not only prove Property 7.2.6 but also show how to find such a tight set in polynomial time.

Given these properties, we can augment $\mathcal{C}$ as follows: (1) For every $A \in \mathcal{C}$ such that $Q \subset A$, replace $A$ by $Q'' \cup A$. (2) Add $Q'$ and $Q''$ to $\mathcal{C}$. (3) Find set $W$ satisfying Property 7.2.6 and add $W$ to $\mathcal{C}$. These three steps maintain the invariant that $\mathcal{C}$ is a chain of tight subsets of $S'$, and they increase $|\mathcal{C}| - |S'|$ by one.

## 7.3 Proof of Correctness

The main goal of this section is to prove the following theorem, which restates Property 7.2.6:

**Theorem 7.3.1** There exists a tight set $W$ in $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$ such that $Q'' \subset W \subset (Q'' \cup R)$, and $W$ is computable in $O(\text{FEAS})$ time.

After proving this theorem, the correctness and running time of our dynamic transshipment algorithm quickly follow. First, however, we prove several lemmas concerning any subset of terminals $A \subseteq S'$. Let functions $o$ and $v$ to refer to feasible dynamic transshipment problem $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$, while $o'$ and $v'$ refer to infeasible $(\overline{\mathcal{N}}^{\delta^*-1}, \overline{v}^{\delta^*-1}, T)$.

**Lemma 7.3.2** If $o'(A) < v'(A)$ then $o(A) = v(A)$ and $s_i^{j+1} \in A$ but $s_i^0 \notin A$.

**Proof:** Suppose $o'(A) < v'(A)$. Feasibility implies $o(A) \geq v(A)$. Decreasing the delay $\delta$ of source $s_i^{j+1}$ cannot decrease the maximum dynamic flow out of any set: $o'(A) \geq o(A)$. Furthermore, decreasing $\delta$ has no effect on $o(A)$ for any $A \not\ni s_i^{j+1}$; but a unit decrease of $\delta$ could increase $o(A)$ by at most one if $s_i^{j+1} \in A$. Applied to $Q'$ and $Q''$, this means $v_{s_i^{j+1}} \leq v'_{s_i^{j+1}} \leq v_{s_i^{j+1}} + 1$. Combined with the observation that no terminal supply other than $v_{s_i^{j+1}}$ can increase, the above inequalities yield

$$v'(A) > o'(A) \geq o(A) \geq v(A) \geq v'(A) - 1.$$

Notice that all but the first element of this chain must in fact be equal, so that $A$ is a tight set: $o(A) = v(A)$. This chain also shows $v'(A) > v(A)$; because parameter $\delta$ changes $v_{s_i^0}$ exactly opposite to $v_{s_i^{j+1}}$, this means that $s_i^{j+1} \in A$ but $s_i^0 \notin A$. ■

**Lemma 7.3.3** If $(A \cap R) \subseteq Q''$ then $o'(A) \geq v'(A)$.

**Proof:** We prove the lemma in two parts:

**(1)** *If $A \subseteq Q''$ then $o'(A) \geq v'(A)$.*

Suppose $A \subseteq Q''$. By Lemma 7.3.2, if $s_i^{j+1} \notin A$ then $o'(A) \geq v'(A)$; thus we need only consider the case when $s_i^{j+1} \in A$, which means $A \cup Q' = Q''$. Using submodularity with $A$ and $Q'$, we get $o'(A) \geq o'(Q'') + o'(A \cap Q') - o'(Q')$. Both $Q'$ and $Q''$ are tight. Since $s_i^{j+1} \notin A \cap Q'$, Lemma 7.3.2 implies $o'(A) \geq v'(Q'') + v'(A \cap Q') - v'(Q') = v'(A)$.

**(2)** *If $(A \cap R) \subseteq Q''$ then $o'(A) \geq v'(A)$.*

Let $R'' = (Q'' \cup R)$. Suppose $(A \cap R) \subseteq Q''$; this implies $(A \cap R'') \subseteq Q''$. Using submodularity with $A$ and $R''$, we get $o'(A) \geq o'(A \cup R'') + o'(A \cap R'') - o'(R'')$. Since $s_i^0 \in (A \cup R'')$, Lemma 7.3.2 implies $o'(A \cup R'') \geq v'(A \cup R'')$. Because we assume $(A \cap R'') \subseteq Q''$, the first part of the proof implies $o'(A \cap R'') \geq v'(A \cap R'')$. Property 7.2.5 implies $o'(R'') = v'(R'')$. Thus, we obtain $o'(A) \geq v'(A \cup R'') + v'(A \cap R'') - v'(R'') = v'(A)$. ■

The following Lemma 7.3.4 is a well-known property of tight sets that follows from the submodularity of function $o$:

**Lemma 7.3.4** In a feasible dynamic transshipment problem, the union and intersection of tight sets are tight.

We are now ready to prove Theorem 7.3.1:

**Proof of Theorem 7.3.1:** Let $W'$ be any violated set in infeasible problem $(\overline{\mathcal{N}}^{\delta^*-1}, \overline{v}^{\delta^*-1}, T)$. By Lemma 7.3.2, $W'$ is a tight set in $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$. Lemma 7.3.2 also implies $s_i^0 \notin W'$. Because $s_i^0 \in R \backslash Q$, this means $R \not\subseteq (Q'' \cup W')$. Lemma 7.3.3 implies $(W' \cap R) \not\subseteq Q''$.

Consider $W = Q'' \cup (W' \cap R)$. The above statements imply that $Q'' \subset W \subset (Q'' \cup R)$. Furthermore, after rewriting $W$ as $(Q'' \cup W') \cap (Q'' \cup R)$, observe that Properties 7.2.3 and 7.2.5 and Lemma 7.3.4 imply that $W$ is a tight set in $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$.

To prove the running time, observe that $W'$ is an arbitrary violated set whose computation requires only one feasibility test of $(\overline{\mathcal{N}}^{\delta^*-1}, \overline{v}^{\delta^*-1}, T)$. ■

**Theorem 7.3.5** The dynamic transshipment problem can be solved in $O(k \log(nUT)\mathrm{FEAS})$ time.

**Proof:** The algorithm is dominated by the $O(k)$ iterations described in Section 7.2.2. Each iteration is dominated by two binary searches. The first $\alpha$-binary search tests feasibility each time it seeks integer $\alpha^* \in [0, nU]$. The second $\delta$-binary search tests feasibility each time it seeks integer $\delta^* \in [0, T+1]$. ■

**Theorem 7.3.6** The dynamic transshipment problem can be solved in strongly polynomial time.

**Proof:** We proved in Corollary 7.1.17 that dynamic transshipment feasibility can be tested in strongly polynomial time. Let $\mathcal{A}$ be such a strongly polynomial algorithm; suppose this algorithm consists of $p$ comparisons and $q$ additions, where $p$ and $q$ depend on the number of nodes and edges in the network. Using the parametric search technique of Megiddo [38], we can replace each binary search in our polynomial algorithm by a strongly polynomial subroutine consisting of $O(p(q+p))$ operations. The result is a dynamic transshipment algorithm running in $O(kp(q+p))$ time. We sketch Megiddo's technique in the remark below. ■

**Remark on Parametric Search.** Megiddo [38] described parametric search. Suppose we have a strongly polynomial algorithm $\mathcal{A}$ to test feasibility of dynamic transshipment problems; each step of algorithm $\mathcal{A}$ consists only of additions, scalar multiplications, and comparisons. We modify algorithm $\mathcal{A}$ to test feasibility of parameterized dynamic transshipment problems, where the input includes one linear parameter $\lambda$. Each scalar value $a_i$ considered by $\mathcal{A}$ corresponds to a linear function $a_i + \lambda b_i$ in the parameterized problem. Instead of adding scalars $a_i + a_j$, the modified algorithm adds linear functions $(a_i + \lambda b_i) + (a_j + \lambda b_j)$. This modification (and that for scalar multiplication) does not change the big-Oh running time of the algorithm; however, parameterized comparisons are more difficult. We cannot distinguish the possible relationships of $(a_i + \lambda b_i)$ and $(a_j + \lambda b_j)$ without some information about $\lambda$ (unless $b_i = b_j$). All we need to know about $\lambda$, however, is its relationship to the critical value $\lambda^*$, determined by $(a_i + \lambda^* b_i) = (a_j + \lambda^* b_j)$.

In both the $\alpha$-capacity and $\delta$-length binary searches, we have a network with one linear parameter. In both cases, we can determine whether $\lambda \leq \lambda^*$ or $\lambda \geq \lambda^*$ by running the original algorithm $\mathcal{A}$ on the non-parameterized network that substitutes critical value $\lambda^*$ for parameter $\lambda$. With that information in hand, the modified algorithm can resume computing on the parameterized network. An improved (sequential) strongly polynomial algorithm may be derived by a similar technique of Megiddo [39] that uses parallel algorithms to solve parametric optimization problems efficiently.

# Chapter 8

# Extensions and Applications

In Section 8.1, we describe how to solve the dynamic transshipment problem in a network of edges that only admit flow during specified time intervals. In Section 8.2, we solve the quickest transshipment problem, a version of the dynamic transshipment problem in which the time horizon $T$ is not specified but must be minimized. In Section 8.3, we apply our quickest transshipment algorithm to solve a problem of scheduling unit-size jobs on a network of computers.

## 8.1 Dynamic Dynamic Network Flows

One could argue that the "dynamic networks" we consider in this thesis are not particularly "dynamic." Although the flow in a dynamic network does change with time, we assume that the dynamic network itself does not change at all; it is essentially "static." In this section, we discuss some enhancements to traditional dynamic networks that allow the characteristics of a network to change with time while still yielding to the efficient algorithms of this thesis.

Our enhancements culminate with the idea of *mortal edges.* A mortal edge $yz$ in $E^+$ is characterized not only by a capacity $u_{yz}$ and transit time $\tau_{yz}$ but also by a start time $\alpha_{yz}$ and an end time $\beta_{yz}$. Mortal edge $yz$ cannot admit flow outside the time interval $[\alpha_{yz}, \beta_{yz}]$; however, we allow any node to hold an arbitrary non-negative amount of flow at any time. We show how to solve the dynamic transshipment problem in a network of mortal edges in polynomial time. (Note that if we allow mortal edges but not holdover flow, then Klinz [33] observed that the problem becomes NP-hard.) By connecting nodes with parallel mortal edges, we can model problems with time-varying edge capacities and transit times; unfortunately, algorithm performance suffers unless the time-varying edge characteristics remain constant most of the time.

**Terminal Restrictions.** In a traditional dynamic transshipment problem, flow may leave any source node starting at time zero. We can generalize the problem by specifying a release time vector; each source $s_i$ cannot send flow before its release time $r_{s_i}$. This problem reduces to the original version simply by adding to the network a new set of sources $S'$; each new source $s'_i$ in $S'$ corresponds to an old source $s_i$ in $S$; they are connected by an uncapacitated edge $s'_i s_i$ with transit time $r_{s_i}$. In this augmented network, allowing flow to leave $s'_i$ at time zero is equivalent to holding flow at $s_i$ until its release time $r_{s_i}$. In a similar manner, we can also allow deadlines for sinks, so that each sink $s_i$ cannot receive flow after its specified deadline $d_{s_i}$. In this case we connect a new sink $s'_i$ to the network by uncapacitated edge $s_i s'_i$ with transit time $T - d_{s_i}$. We can also limit the rate at which sources can send flow or sinks can receive flow by augmenting the network as above but with capacitated edges.

**Mortal Edges.** We can solve dynamic transshipment problems on networks built of mortal edges by reducing each mortal edge to a small set of traditional edges. Throughout our discussion, we focus on positive-capacity mortal edges in $E^+$. For each mortal edge $yz$ in $E^+$, we assume $0 \leq \alpha_{yz} \leq \beta_{yz} \leq T$. Mortal edge $yz$ requires a new source $s^+_{yz}$ with release time $\alpha_{yz}$ and a new sink $s^-_{yz}$ with deadline $\beta_{yz}$; terminal $s^+_{yz}$ (or $s^-_{yz}$) must send (or receive) $u_{yz}(\beta_{yz} - \alpha_{yz} + 1)$ total units of flow at a rate no more than $u_{yz}$ per time step. Mortal edge $yz$ then reduces to

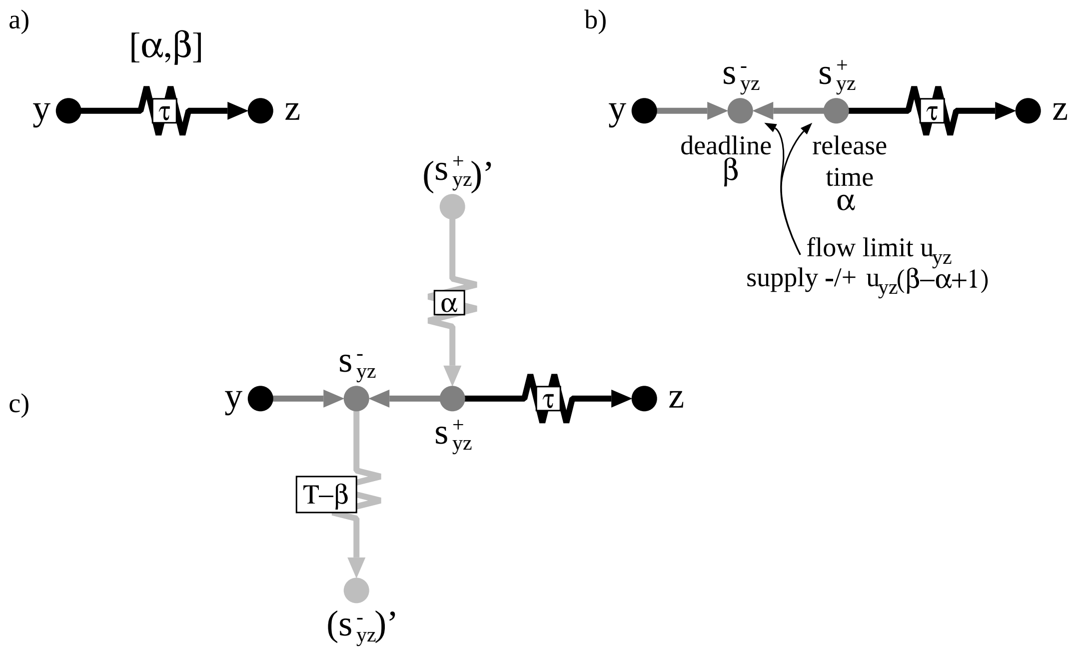


Figure 8.1: Reduction of a Mortal Edge to a Traditional Network

three traditional edges: $ys^-_{yz}$, $s^+_{yz}s^-_{yz}$, and $s^+_{yz}z$. The capacity of each edge is $u_{yz}$. Edge $s^+_{yz}z$ has transit time $\tau_{yz}$, while both $ys^-_{yz}$ and $s^+_{yz}s^-_{yz}$ have length zero. The supplies of $y$ and $z$ remain unchanged.

Figures 8.1(a,b) depict the reduction of mortal edge $yz$ to a traditional network with release times, deadlines, and flow limits. This reduction is very similar to the standard technique of transforming a capacitated minimum-cost flow problem into an uncapacitated minimum-cost flow problem. Figure 8.1(c) further reduces the network to use terminals without release times, deadlines, or flow limits.

Let $\mathcal{D}$ be a dynamic transshipment problem with mortal edges. The above reduction yields a dynamic transshipment problem $\mathcal{D}'$ with release times, deadlines, and terminal flow limits, but without mortal edges. We prove that these two problems are equivalent, in that a solution to one easily yields a solution to the other. The first direction is trivial to check:

**Lemma 8.1.1** Suppose flow $f$ solves dynamic transshipment problem $\mathcal{D}$. Then $\mathcal{D}'$ is solved by flow $f'$, defined by

$$\begin{aligned} f'_{ys^-_{yz}}(\theta) &= f_{yz}(\theta) \\ f'_{s^+_{yz}s^-_{yz}}(\theta) &= u_{yz} - f_{yz}(\theta) \\ f'_{s^+_{yz}z}(\theta) &= f_{yz}(\theta) \end{aligned}$$

for all $yz \in E^+$, $\theta \in [\alpha_{yz}, \beta_{yz}]$. The rest of $f'$ is zero except as determined by antisymmetry.

Next, we prove that a solution to problem $\mathcal{D}'$ yields a solution to problem $\mathcal{D}$. (We claim without proof that we can solve problem $\mathcal{D}'$, since the further reduction of $\mathcal{D}'$ to a problem with no release times, deadlines, or flow limits is trivial.) We need the following lemma concerning any solution to $\mathcal{D}'$:

**Lemma 8.1.2** Suppose flow $f'$ solves dynamic transshipment problem $\mathcal{D}'$. For any edge $yz$ in $E^+$ and any time $\theta$:

$$\begin{aligned} \theta < \alpha_{yz} &\Rightarrow |f'_{s^+_{yz}}|_\theta = 0 \\ \theta \geq \alpha_{yz} &\Rightarrow |f'_{s^+_{yz}}|_\theta \leq u_{yz}(\theta - \alpha_{yz} + 1) \\ \theta \leq \beta_{yz} &\Rightarrow -|f'_{s^-_{yz}}|_\theta \geq u_{yz}(\theta - \alpha_{yz} + 1) \\ \theta > \beta_{yz} &\Rightarrow -|f'_{s^-_{yz}}|_\theta = u_{yz}(\beta_{yz} - \alpha_{yz} + 1). \end{aligned}$$

**Proof:** The first two constraints follow from the release time $\alpha_{yz}$ of source $s^+_{yz}$; before this time $s^+_{yz}$ can send no flow, and afterward it can send no more than $u_{yz}$ per time step. The last two constraints follow from the deadline $\beta_{yz}$ of sink $s^-_{yz}$; before this time $s^-_{yz}$ must have received enough flow so that an additional $u_{yz}$ per remaining time step will meet its demand, and afterward it must be sated. ■

**Lemma 8.1.3** Suppose flow $f'$ solves dynamic transshipment problem $\mathcal{D}'$. Then $\mathcal{D}$ is solved by flow $f$, defined by

$$\begin{aligned} f_{yz}(\theta) &= u_{yz} - f'_{s^+_{yz}s^-_{yz}}(\theta) & \forall\, yz \in E^+,\ \theta \in [\alpha_{yz}, \beta_{yz}] \\ f_{yz}(\theta) &= 0 & \forall\, yz \in E^+,\ \theta \notin [\alpha_{yz}, \beta_{yz}]. \end{aligned}$$

**Proof:** First we prove that $f$ satisfies terminal supplies. This follows from the fact that $f'$ satisfies supplies plus the observation that the total supply of $s^+_{yz}$ and $s^-_{yz}$ is zero (which means $|f_{yz}|_T = u_{yz}(\beta_{yz} - \alpha_{yz} + 1) - |f'_{s^+_{yz}s^-_{yz}}|_T = |f'_{ys^-_{yz}}|_T = |f'_{s^+_{yz}z}|_T$) for any edge $yz$ in $E^+$.

Next, we prove that $f$ is feasible. Dynamic flow $f$ must obey antisymmetry, conservation, capacity, and mortality constraints. The antisymmetry constraints are trivial. For any node $y$, conservation constraint (2.4) follows from the same argument we made to prove that $f$ satisfies terminal supplies. The capacity constraint of any edge $yz$ in $E^+$ follows from the feasibility of flow $f'$. Our definition of $f$ guarantees edge mortality.

The only remaining constraint is conservation constraint (2.2). We must check that the holdover flow is always non-negative at every node. (This is a capacity constraint in the time-expanded network.) We prove that $|f_y|_\theta = \sum_{yz\in E^+} |f_{yz}|_\theta - \sum_{xy\in E^+} |f_{xy}|_\theta \leq 0$ for any node $y$ and time $\theta$; we do so by proving $|f_{xy}|_\theta \geq |f'_{s^+_{xy}y}|_\theta$ and $|f_{yz}|_\theta \leq |f'_{ys^-_{yz}}|_\theta$ for any edges $xy, yz$ in $E^+$. These inequalities together with the feasibility of $f'$ and the identical transit times of $xy$ and $s^+_{xy}y$ imply the lemma.

**(1)** $|f_{xy}|_\theta \geq |f'_{s^+_{xy}y}|_\theta$.

Consider any edge $xy$ in $E^+$. If $\theta < \alpha_{xy}$ then $|f_{xy}|_\theta = 0 = |f'_{s^+_{xy}y}|_\theta$. If $\theta > \beta_{xy}$ then $|f_{xy}|_\theta = |f_{xy}|_T = |f'_{s^+_{xy}y}|_T \geq |f'_{s^+_{xy}y}|_\theta$. Otherwise, $|f_{xy}|_\theta = u_{xy}(\theta - \alpha_{xy} + 1) - |f'_{s^+_{xy}s^-_{xy}}|_\theta$. By Lemma 8.1.2, this is at least $|f'_{s^+_{xy}}|_\theta - |f'_{s^+_{xy}s^-_{xy}}|_\theta = |f'_{s^+_{xy}y}|_\theta$.

**(2)** $|f_{yz}|_\theta \leq |f'_{ys^-_{yz}}|_\theta$.

Consider any edge $yz$ in $E^+$. If $\theta < \alpha_{yz}$ then $|f_{yz}|_\theta = 0 \leq |f'_{ys^-_{yz}}|_\theta$. If $\theta > \beta_{yz}$ then $|f_{yz}|_\theta = |f_{yz}|_T = |f'_{ys^-_{yz}}|_T = |f'_{ys^-_{yz}}|_\theta$. Otherwise, $|f_{yz}|_\theta = u_{yz}(\theta - \alpha_{yz} + 1) - |f'_{s^+_{yz}s^-_{yz}}|_\theta$. By Lemma 8.1.2, this is no more than $-|f'_{s^-_{yz}}|_\theta - |f'_{s^+_{yz}s^-_{yz}}|_\theta = |f'_{ys^-_{yz}}|_\theta$. ■

## 8.2 The Quickest Transshipment Problem

In the *quickest transshipment problem*, we are given a dynamic network $\mathcal{N}$ and supply vector $v$. We seek a feasible dynamic flow that satisfies supply $v$ within the minimum possible time horizon, if such a flow exists. We denote an instance of the quickest transshipment problem by the pair $(\mathcal{N}, v)$.

To solve a quickest transshipment problem $(\mathcal{N}, v)$, we must find the minimum time $T$ such that the dynamic transshipment problem $(\mathcal{N}, v, T)$ is feasible. This can be done in polynomial time using a binary search and Theorem 7.1.12. To analyze the binary search, we must bound the optimal time horizon $T$ in terms of the input network $\mathcal{N}$ and supply vector $v$. Recall that $\mathcal{T}_{\mathrm{m}}$ is the maximum transit time in network $\mathcal{N}$, and $V$ denotes the total supply of all sources $v(S^+)$.

**Lemma 8.2.1** For any quickest transshipment problem $(\mathcal{N}, v)$, the optimal time horizon $T$ is no more than $n\mathcal{T}_{\mathrm{m}} + V$.

**Proof:** Theorem 7.1.1 implies that there is some terminal subset $W \subset S$ such that the quickest flow sending $v(W)$ units of flow collectively from the sources of $W$ to the sinks of $S \setminus W$ takes $T$ time steps; this is equivalent to a single-source single-sink quickest flow. An upper bound for this simpler problem is therefore also an upper bound for the quickest transshipment problem.

Consider the first time step when flow reaches a sink; this must happen by time $n\mathcal{T}_{\mathrm{m}}$. In a single-source single-sink setting, it is easy to see that after $n\mathcal{T}_{\mathrm{m}}$ time steps, at least one unit of flow can reach the sink with each time step, until there is no more flow. Thus the optimal time horizon for the quickest transshipment problem cannot be more than $n\mathcal{T}_{\mathrm{m}} + V$. ■

**Corollary 8.2.2** The optimal time horizon $T$ for the quickest transshipment problem can be found in $O(\log(nV\mathcal{T}_{\mathrm{m}})\mathrm{FEAS})$ time and also in strongly polynomial time.

For the strongly polynomial time bound, we use Megiddo's parametric search [38] instead of binary search. We sketched Megiddo's technique after the proof of Theorem 7.3.6.

Given the optimal time horizon $T$ for a quickest transshipment problem $(\mathcal{N}, v)$, we have a dynamic transshipment problem that we know how to solve efficiently by Theorem 7.3.5 or 7.3.6. Notice that the running time we need to determine $T$ is dominated by the running time we need to solve the dynamic transshipment problem $(\mathcal{N}, v, T)$. Thus, our quickest transshipment algorithm requires the same running time as our dynamic transshipment algorithm, except that we use Lemma 8.2.1 to express the bound in terms of $\mathcal{N}$ and $v$ but not $T$.

**Corollary 8.2.3** The quickest transshipment problem (and the dynamic transshipment problem) can be solved in $O(k \log(nUV\mathcal{T}_{\mathrm{m}})\mathrm{FEAS})$ time and also in strongly polynomial time.

## 8.3 Network Scheduling

We introduced network scheduling in Chapters 1 and 3. In this section, we consider the unit-size-job network scheduling problem and show how it reduces to the quickest transshipment problem. Our quickest transshipment algorithm is the first efficient algorithm for general unit-size-job network scheduling.

**Theorem 8.3.1** The unit-size-job network scheduling problem can be solved via one quickest transshipment computation.

**Proof:** Consider an instance of the unit-size-job network scheduling problem. Let $V$ be the set of processors, connected by a set of directed links $E$, which are characterized by capacity and transit time functions $u$ and $\tau$. Let $v_x$ be the number of unit-size jobs initially assigned to each processor $x$ in $V$.

We reduce this to an integral quickest transshipment problem by adding a supersink $t$ to the network. Connect each processor $x$ in $V$ to supersink $t$ by a unit-capacity zero-transit-time edge $xt$. Let $V'$ and $E'$ denote the augmented graph, and likewise denote the extended capacity and transit time functions by $u'$ and $\tau'$. We define a supply vector $v'$ based on $v$ as follows: $v'_x = v_x$ for all nodes $x$ in $V$, and $v'_t = -v(V)$. Network $((V', E'), u', \tau', V')$ with supply vector $v'$ is an integral quickest transshipment problem. A solution to this dynamic flow problem corresponds directly to an optimal network schedule — each unit of flow corresponds to one unit-size job, and sending one unit of flow from some node $x$ in $V$ to supersink $t$ corresponds to executing one job on processor $x$. ■

Notice in the above proof that we can also model faster processors. If each processor $x$ in $V$ can execute $s_x$ jobs per time step (where $s_x$ is a non-negative integer), then in our reduction each edge $xt$ entering the supersink has capacity $s_x$.

Other groups have obtained polynomial algorithms for unit-job network scheduling, but only for very special cases. Deng *et al* [13] considered networks with unit transit times and no capacities. Fizzano and Stein [15] considered ring networks with unit transit times and unit capacities. There are many other varieties of network scheduling problems and some interesting open questions for research in this area; we sketch some of these in the next chapter.

# Chapter 9

# Summary and Open Problems

Current theory and practice in the field of dynamic network flows relies overwhelmingly on time-expanded networks. We introduce chain-decomposable flows in this thesis as an alternative to the traditional technique. Chain-decomposable flows lead to the first polynomial algorithms for several important dynamic network flow problems: notably, the quickest transshipment problem, the dynamic transshipment problem, and the lexicographically maximum dynamic flow problem. We also use chain-decomposable flows to compute universally maximum dynamic flows; this approach leads to the first polynomial approximation algorithm for this problem and the first polynomial algorithm to compute a time-step snapshot of a universally maximum dynamic flow.

Chain-decomposable flows generalize two previously known techniques used in efficient dynamic network flow algorithms: stationary dynamic flows and standard chain-decomposable flows. These techniques have proven useful for solving infinite-horizon problems and the maximum dynamic flow problem, but little else. The structure of chain-decomposable flows is flexible enough to support polynomial algorithms for a much wider range of problems.

In the rest of this chapter, we discuss some interesting open problems related to dynamic network flows.

**Combinatorial vs Geometric Algorithms.** The inception of the field of network flows coincided with the development of optimization in general. Network flow problems were quickly recognized as a special type of linear program that is amenable to solution by the simplex method and other general techniques for optimizing linear objective functions over polyhedra. Since Ford and Fulkerson, however, progress in network flow theory has been almost exclusively combinatorial, relying more on the special structure of networks and less on the general structure of polyhedra. An intriguing result of this thesis is the reunification of general polyhedral theory with state-of-the-art network flow algorithms. For the moment, we test dynamic transshipment feasibility with a geometric algorithm that can optimize an arbitrary linear objective function over a convex set.

It is an interesting open problem to develop a purely combinatorial polynomial dynamic transshipment algorithm. A successful investigation might yield a more efficient algorithm (theoretically) and would almost certainly yield a more practical one. There are two intriguing lines of attack on this problem. First, geometric methods ignore the network structure inherent to dynamic transshipment problems. By considering a direct approach that uses network structure, one might obtain a better algorithm for feasibility testing. Furthermore, it seems likely that such an algorithm would not only decide feasibility but also compute a solution when given a feasible dynamic transshipment problem. Because our current dynamic transshipment algorithm relies on repeated feasibility testing in order to compute a solution, replacing our geometric feasibility algorithm with a network-based approach could yield even bigger improvements to our dynamic transshipment algorithm. Second, Queyranne discovered a simple combinatorial algorithm to minimize *symmetric* submodular functions [49]. Even in a dynamic network consisting only of "two-way streets" (edges $yz$ and $zy$ always paired with identical capacities and transit times) we cannot directly apply his result; however, there may be some way to generalize Queyranne's approach to our case.

**Multicommodity Dynamic Network Flows.** Multicommodity flows model the transportation of several distinct types of flow through a single network; each commodity is supplied by its own set of sources and demanded by its own set of sinks. Multicommodity flow problems are typically much harder than their single-commodity counterparts. In a traditional network without transit times, a multicommodity flow problem can be formulated as a linear program and solved in polynomial time, provided that a fractional solution is allowed. If integral flows are required, then even the two-commodity flow problem is NP-complete; research in this area has focused on approximation algorithms.

In a dynamic network, integral multicommodity flows are better known as packet-routing, a topic of great interest. The tremendous difficulty and vital importance of this problem have spurred a huge research effort. Flow theory has not played much of a role. Research in this area typically involves heuristics with empirical support, stochastic models with queuing theory, or deterministic models with distributed on-line approximation algorithms.

It is an interesting open problem to develop a polynomial algorithm to solve fractional multicommodity dynamic flow problems. Like their static counterparts, fractional multicommodity dynamic flow problems can be formulated as linear programs. In the dynamic case, however, the linear program corresponds to an exponentially large time-expanded network, and so existing linear program solvers do not suffice. One needs to take advantage of the fact that, despite its exponential number of variables and constraints, this linear program has a simple underlying structure.

**Minimum-Cost Dynamic Network Flows.** In the quickest transshipment problem and its multicommodity extension, we know how long it takes flow to get through each edge, and we want to drain the last unit of flow from the network within some overall time bound. It is natural to extend these problems by considering also the cost of sending flow through each edge; then we want to drain the network not only within some overall time bound, but also within an overall cost constraint, or budget. Unfortunately, Klinz and Woeginger [34] showed that controlling both time and cost simultaneously leads to many NP-hard problems, even in networks where controlling time or cost separately is trivial.

Suppose we model the roads of greater Los Angeles: the cost might be twenty-five cents per mile, while the transit time is three minutes per mile. Each commuter lives at a source node in the network and works at a sink node in the network. Solving this minimum-cost multicommodity dynamic flow problem would indicate how to route morning rush hour traffic so that the longest commute is as short as possible and the total transportation costs are minimized.

The scenario above leads naturally to a special class of dynamic networks in which the cost of each edge is proportional to its transit time. Given networks in this class, we do not know minimum-cost dynamic network flow problems to be NP-hard; unfortunately, we do not know any polynomial algorithms for these problems either. We do not even know how to approximate dynamic flows with time and budget constraints. It is an interesting open problem to consider (even single-commodity) minimum-cost dynamic flow problems given proportional edge costs and transit times.

**Preemptive Network Scheduling.** A *non-preemptive* schedule can interrupt a job but can only restart that job at the beginning again; a *preemptive* schedule is allowed to interrupt a job during execution and then restart it where it left off. This distinction is pointless in the unit-size-job network scheduling problem because there is no reason to interrupt one unit-size job for another (assuming all data are integral). When job sizes are not uniform, however, then some preemptive schedules may execute faster than any non-preemptive schedule. All the results of Phillips *et al* [46] apply to non-preemptive scheduling (with no capacities). Despite the fact that many systems schedule jobs preemptively, virtually nothing is known about preemptive network scheduling. Rayward-Smith [50] proved that preemptive network scheduling is NP-hard, even when the network is a clique and all interprocessor delays are two (or more). It is an interesting open problem to find a good approximation algorithm for general preemptive scheduling in networks with capacities and transit times.

# Appendix A

# Epilogue

Jack and Rachel relaxed aboard the *Manetho*, a nearby cargo ship that had picked up the *Combitanic*'s distress call and recovered her passengers.

"That was wonderful, Jack," Rachel cooed. "But I have one question: did you actually implement one of the polynomial quickest transshipment algorithms, or did you just use exhaustive search to compute all the tight sets of cabins on the *Combitanic*?"

"I think that's enough of my secrets for now," Jack said. "My implementation is proprietary. What *I* want to know is, who was that drunken sailor I found you with?"

"Dirk Crandall... but he's gone, Jack. You're the only man for me. I love you maximally."

THE END

# Bibliography

[1] R.K. Ahuja, A.V. Goldberg, J.B. Orlin, and R.E. Tarjan. Finding minimum-cost flows by double scaling. *Mathematical Programming*, 53:243–266, 1992.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1993.

[3] E.J. Anderson and A.B. Philpott. A continuous-time network simplex algorithm. *Networks*, 19:395–425, 1989.

[4] J.E. Aronson. A survey of dynamic network flows. *Annals of Operations Research*, 20:1–66, 1989.

[5] G.N. Berlin. The use of directed routes for assessing escape potential, 1979. National Fire Protection Association, Boston, MA.

[6] R.E. Burkard, K. Dlaska, and H. Kellerer. The quickest disjoint flow problem. Technical Report 189-91, Institute of Mathematics, University of Technology, Graz, Austria, 1991.

[7] R.E. Burkard, K. Dlaska, and B. Klinz. The quickest flow problem. *ZOR Methods and Models of Operations Research*, 37(1):31–58, 1993.

[8] L.G. Chalmet, R.L. Francis, and P.B. Saunders. Network models for building evacuation. *Management Science*, 28:86–105, 1982.

[9] Y.L. Chen and Y.H. Chin. The quickest path problem. *Computers and Operations Research*, 17:153–161, 1990.

[10] W. Choi, R.L. Francis, H.W. Hamacher, and S. Tufecki. Network models of building evacuation problems with flow-dependent exit capacities. In J.P. Brans, editor, *Proc. 10th Int. Conf. on Operations Research*, pages 1047–1059. North-Holland, Amsterdam, August 1984.

[11] W. Choi, H.W. Hamacher, and S. Tufecki. Modeling of building evacuation problems by network flows with side constraints. *Eur. J. Operations Research*, 35:98–110, 1988.

[12] E. Cohen and N. Megiddo. Strongly polynomial-time and NC algorithms for detecting cycles in dynamic graphs. In *Proc. 21st Annual ACM Symp. on Theory of Computing*, pages 523–534, 1989.

[13] X. Deng, H. Liu, and B. Xiao. Deterministic load balancing in computer networks. In *Proc. 2nd IEEE Symp. on Parallel and Distributed Processing*, pages 50–57, 1992.

[14] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and their Applications*, pages 69–87. Gordon and Breach, 1970.

[15] P. Fizzano and C. Stein. Scheduling on a ring with unit capacity links. Technical Report PCS-TR94-216, Dartmouth College, 1994.

[16] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, New Jersey, 1962.

[17] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal for the Assoc. for Comp. Mach.*, 34(3):596–615, 1987.

[18] D. Gale. Transient flows in networks. *Michigan Mathematical Journal*, 6:59–63, 1959.

[19] G. Gallo, M.D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal of Computing*, 18(1):30–55, 1989.

[20] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

[21] A.V. Goldberg and R.E. Tarjan. Solving minimum-cost flow problems by successive approximation. *Mathematics of Operation Research*, 15:430–466, 1990.

[22] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, 1988.

[23] B. Hajek and R.G. Ogier. Optimal dynamic routing in communication networks with continuous traffic. *Networks*, 14:457–487, 1984.

[24] J. Halpern. A generalized dynamic flows problem. *Networks*, 9:133–167, 1979.

[25] H.W. Hamacher. Temporally repeated flow algorithms for dynamic min cost flows. In *Proc. 28-th IEEE Conference on Decision and Control*, pages 1142–1146, 1989.

[26] H.W. Hamacher and S. Tufekci. On the use of lexicographic min cost flows in evacuation modeling. *Naval Research Logistics*, 34:487–503, 1987.

[27] B. Hoppe and É. Tardos. Polynomial time algorithms for some evacuation problems. In *Proc. 5th Annual ACM-SIAM Symp. Discrete Algorithms*, pages 433–441, 1994.

[28] B. Hoppe and É. Tardos. The quickest transshipment problem. In *Proc. 6th Annual ACM-SIAM Symp. Discrete Algorithms*, pages 512–521, 1995.

[29] Y.-C. Hung and G.-H. Chen. On the quickest path problem. In *Proc. ICCI*, pages 44–46. Springer-Verlag, 1991. Lecture Notes in Computer Science 497.

[30] K. Iwano and K. Steiglitz. Testing for cycles in infinite graphs with periodic structure. In *Proc. 19th Annual ACM Symp. on Theory of Computing*, pages 46–55, 1987.

[31] J.R. Jarvis and D.H. Ratliff. Some equivalent objectives for dynamic network flow problems. *Management Science*, 28:106–109, 1982.

[32] D. Kagaris, G.E. Pantziou, S. Tragoudas, and C.D. Zaroliagis. On the computation of fast data transmissions in networks with capacities and delays. In *Proc. Workshop on Algorithms and Data Structures*, 1995. (To appear).

[33] B. Klinz. Personal communication.

[34] B. Klinz and G.J. Woeginger. Minimum cost dynamic flows: The series-parallel case. In E. Balas and J. Clausen, editors, *Proc. 4th Conference on Integer Programming and Combinatorial Optimization*, pages 329–343. Springer-Verlag, 1995. Lecture Notes in Computer Science 920.

[35] S.R. Kosaraju and G. Sullivan. Detecting cycles in dynamic graphs in polynomial time. In *Proc. 20th Annual ACM Symp. on Theory of Computing*, pages 398–406, 1988.

[36] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S.C. Graves *et al*, editor, *Handbooks in Operations Research and Management Science, Vol. 4*. Elsevier Science Publishers B.V., 1993.

[37] N. Megiddo. Optimal flows in networks with multiple sources and sinks. *Mathematical Programming*, 7:97–107, 1974.

[38] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.

[39] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

[40] E. Minieka. Maximal, lexicographic, and dynamic network flows. *Operations Research*, 21:517–527, 1973.

[41] E. Minieka. Dynamic network flows with arc changes. *Networks*, 4:255–265, 1974.

[42] J.B. Orlin. Maximum-throughput dynamic network flows. *Mathematical Programming*, 27:214–231, 1983.

[43] J.B. Orlin. Minimum convex cost dynamic network flows. *Mathematics of Operations Research*, 9:190–207, 1984.

[44] J.B. Orlin. Some problems on dynamic/periodic graphs. In W.R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 273–293. Academic Press, Orlando, FL, 1984.

[45] J.B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[46] C. Phillips, C. Stein, and J. Wein. Task scheduling in networks. In *Proc. Fourth Scandinavian Workshop on Algorithm Theory*, pages 290–301, 1994.

[47] W.B. Powell, P. Jaillet, and A. Odoni. Stochastic and dynamic networks and routing. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Handbooks in Operations Research and Management Science: Networks*. Elsevier Science Publishers B.V., 1995.

[48] M.C. Pullan. An algorithm for a class of continuous linear programs. *SIAM Journal on Control and Optimization*, 33(6):1558–1577, 1993.

[49] M. Queyranne. A combinatorial algorithm for minimizing symmetric submodular functions. In *Proc. 6th Annual ACM-SIAM Symp. Discrete Algorithms*, pages 98–101, 1995.

[50] V.J. Rayward-Smith. The complexity of preemptive scheduling given interprocessor communication delays. *Inf. Proc. Letters*, 25:123–125, 1987.

[51] J.B. Rosen, S.-Z. Sun, and G.-L. Xue. Algorithms for the quickest path problem and the enumeration of quickest paths. *Computers and Operations Research*, 18:579–584, 1991.

[52] P.M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proc. 30th Annual IEEE Symp. on Foundations of Computer Science*, pages 338–343, 1989.

[53] B. Veltman. *Multiprocessor Scheduling with Communication Delays*. Ph.D. dissertation, CWI, Amsterdam, 1993.

[54] W.L. Wilkinson. An algorithm for universal maximal dynamic flows in a network. *Operations Research*, 19(7):1602–1612, 1971.

[55] N. Zadeh. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5:255–266, 1973.