The Quickest Transshipment Problem

Bruce Hoppe* Éva Tardos†

Abstract

A dynamic network consists of a graph with capacities and transit times on its edges. The quickest transshipment problem is defined by a dynamic network with several sources and sinks; each source has a specified supply and each sink has a specified demand. The problem is to send exactly the right amount of flow out of each source and into each sink in the minimum overall time.

Variations of the quickest transshipment problem have been studied extensively; the special case of the problem with a single sink is commonly used to model building evacuation. Similar dynamic network flow problems have numerous other applications; in some of these, the capacities are small integers and it is important to find integral flows. There are no polynomial-time algorithms known for most of these problems.

In this paper we give the first polynomial-time algorithm for the quickest transshipment problem. Our algorithm provides an integral optimum flow. Previously, the quickest transshipment problem could only be solved efficiently in the special case of a single source and single sink.

1 Introduction

Transportation modeling spawned the field of network flows in the 1940s. Despite this connection, however, 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?" Parents who endure this persistent question know to consider time when planning a trip.

In this paper we consider dynamic networks, a network flow model that includes transit times on edges. Dynamic network flow problems were introduced by Ford and Fulkerson [14], who considered dynamic networks with a single source and single sink. They showed that a natural variant of the maximum flow problem in single-source single-sink dynamic networks can be solved by one minimum-cost flow computation. In this paper, we describe the first polynomial-time algorithms that solve dynamic network flow problems with multiple sources and/or sinks.

A *dynamic network* is defined by a directed graph $G = (V, E)$ with sources, sinks, and non-negative capacities u_{yz} and integral transit times τ_{yz} for every edge $yz \in E$. We assume a discrete time model in this paper: in a feasible dynamic flow, at most u_{yz} units of flow can enter edge yz with each integral time step, and flow leaving y along edge yz at time θ reaches z at time $\theta + \tau_{yz}$. For example, an edge with capacity 2 and transit time 3 can accept 2 new units of flow at each

*The Sabre Group, 23 Third Ave, Burlington, MA 01803. email: `Bruce_Hoppe@notes.sabre.com`. Research was done while the author was a graduate student at the Department of Computer Science, Cornell University, supported by a National Science Foundation Graduate Research Fellowship.

†Department of Computer Science, Cornell University, Ithaca, NY 14853, email: `eva@cs.cornell.edu`. Research supported in part by a Packard Fellowship, an NSF PYI award, NSF grant CCR-9700163 and ONR grant N00014-98-1-0589.

time step, for a total of up to 6 units of flow in transit on this edge at one time. See Section 2 for a formal definition. In the related continuous-time version, the capacity limits the rate at which flow can enter an edge. See [13] for a comparison of the discrete and continuous models.

The *quickest transshipment problem* is defined by a dynamic network with a set of sources and sinks; each source has a specified supply of flow, and each sink has a specified demand. The problem is to send exactly the right amount of flow out of each source and into each sink in the minimum overall time, if possible. The integral quickest transshipment problem requires a solution that is not only optimal but also integral. (As in traditional network flows, integral instances of dynamic network flow problems always have integer solutions.) This paper describes the first polynomial-time algorithm for the quickest transshipment problem, and also provides an integral flow.

Dynamic networks allow not only flow on edges but also intermediate storage at vertices. This corresponds to holding inventory at a node before sending it onward. We solve the quickest transshipment problem without using such holdover flow, except at the sources and sinks. We prove that our solution is still optimal even if holdover flow is allowed.

Dynamic network flow problems with several sources and sinks arise in many applications (*e.g.*, airline, truck, and railway scheduling). In a number of these applications (*e.g.*, airline scheduling) the capacities in the network are small integers, and it is important to find an integral solution. While there has been a fair amount of work in this area (see the surveys [1, 32]), there are no polynomial-time algorithms known for most of these problems, including the integral quickest transshipment problem with just two sources and one sink.

The *lexicographically maximum dynamic flow problem* is an easier multi-terminal extension of the single-source, single-sink maximum dynamic flow problem. We are given a time horizon T and a dynamic network with an ordered set of terminals (both sources and sinks); we seek a feasible dynamic flow that lexicographically maximizes the amount of flow leaving each terminal in the given order. Note that the solution will have negative or zero flow leaving low-priority terminals. A negative amount $-\alpha$ of flow leaving a terminal means that a positive amount α of flow enters that terminal. Maximizing the amount of flow leaving such a terminal is equivalent to minimizing the amount of flow entering it.

In Section 4, we describe the first polynomial algorithm for the lex max dynamic flow problem. We are interested in the lex max dynamic flow problem primarily because it provides an important subroutine in our polynomial quickest transshipment algorithm.

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*.

The algorithms described here have polynomial or strongly 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 Section 3. All of our algorithms compute chain-decomposable flows.

Dynamic network flows can be 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 at least linearly 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 only logarithmically with finer model granularity, and so may allow much more accurate modeling than traditional techniques.

Research Survey. Special cases of the quickest transshipment problem have been studied elsewhere. The *evacuation problem* is the quickest transshipment problem with a single sink; Berlin [2] and Chalmet *et al* [5] studied this problem as a means of modeling emergency evacuation from buildings. Jarvis and Ratliff [24] 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 [19] 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* [6] studied evacuation given flow-dependent exit capacities; see also [7]. 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.

Much dynamic flow research models time in discrete steps. However, there is also a substantial amount of work on extending results from the discrete models to the continuous-time version, for example see the survey of Anderson [1]. Fleischer [13] and the second author relate the two approaches by extending some of the polynomial-time algorithms that work in the discrete-time model to produce optimal continuous dynamic flows.

Hajek and Ogier [18] considered a continuous-time version of the evacuation problem in the special case when all transit times are zero. They obtained a polynomial-time algorithm by reducing this problem to one maximum flow computation for each source and each sink in the network. 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. Simpler and faster algorithms for this special case were later provided by Fleischer [11] and Fleischer and Orlin [12].

The growing topic of *network scheduling* is closely related to the evacuation problem. A network scheduling problem consists of a set of processors connected by links with capacities and transit times; each processor begins with a set of jobs of various sizes. The goal is to run all the jobs in the minimum overall time, where each job may be run by its home processor or may be sent into the network for remote execution. Based on a simple reduction to the integral evacuation problem (described in Section 8), this paper gives the first polynomial-time algorithm for the network scheduling problem when every job has unit size. Further-restricted special cases have been studied by Deng *et al* [9] and Fizzano and Stein [10]. Deng *et al* [9] described a polynomial-time algorithm for unit-job network scheduling when every link is uncapacitated and has unit transit time. Fizzano and Stein [10] described a polynomial-time algorithm for unit-job network scheduling when every link has unit capacity and unit transit time, and the network is a ring. Phillips *et al* [31] studied network scheduling with arbitrary size jobs but without link capacities. This version of the problem is NP-hard. They obtained an approximation algorithm that produces a schedule no more than twice optimal.

Single-source single-sink variants of the quickest transshipment problem have been well-studied. Two classical problems are the maximum dynamic flow problem and the quickest flow problem. A *maximum dynamic flow* sends as much flow as possible from the source to the sink within a specified time bound T. Ford and Fulkerson [14] showed that the maximum dynamic flow problem can be solved in polynomial time via one minimum-cost flow computation. The *quickest flow problem* is the single-source single-sink version of the quickest transshipment problem. It can be reduced to the maximum dynamic flow problem by binary search; Burkard *et al* [4] gave more efficient and strongly polynomial algorithms for this problem. The *quickest path problem* is a variant of the quickest flow problem that allows only one path for sending flow. The quickest path problem can be solved in polynomial time, as discovered by Chen and Chin [8], Rosen *et al* [34], and Hung and Chen [23]. These results were improved by Kagaris *et al* [25]. Burkard *et al* [3] also described how to compute the quickest flow using h disjoint paths, for fixed h.

All previous research on lexicographically maximum flow problems has focused on static network flows. Minieka [30] studied lex max static flows for the special case in which all sources are ranked higher than all sinks. Megiddo [27] 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* [15] 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 max-flow computation using the preflow-push algorithm of Goldberg and Tarjan [16]. All algorithms for lex max static flows apply to lex max dynamic flows via the exponentially large time-expanded network.

Our results. In this paper we describe the first polynomial-time algorithm that solves dynamic network flow problems with multiple sources and/or sinks. In Section 4 we begin by describing the first polynomial algorithm for the lex max dynamic flow problem. Section 5 describes how to compute the optimal time bound for a quickest transshipment problem. The next two sections contain the main results of this paper: Section 6 shows how to solve a quickest transshipment problem whose time bound is known, by reducing it to a lex max dynamic flow problem; and Section 7 proves the correctness and running time of our algorithm. Finally, Section 8 applies our quickest transshipment algorithm to unit-job network scheduling and also discusses extensions of our main results. Preliminary versions of these results were presented in the proceedings papers [21, 22]. See also the Ph.D. thesis [20] of the first author.

2 Definitions and Preliminaries

A *dynamic network* $\mathcal{N} = (G, u, \tau, S)$ consists of a directed graph $G = (V, E)$ with a non-negative *capacity* u_{yz} and an integral *transit time* τ_{yz} associated with each edge $yz \in E$, and a set of *terminals* $S \subseteq V$. 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 also refer to transit times as *length* and *cost*.

We distinguish edges with positive capacity by including them also in set E^+. The edges in E^+ are the edges in the network used for sending flow. We assume that the edges in E^+ have non-negative transit time. For simplicity of notation, we also assume that E^+ has no parallel or opposite edges, that sources have no entering edges in E^+, and that sinks have no leaving edges in E^+. We include all opposite edges of E^+ in E (so that all residual edges will be in E). For an edge $yz \in E^+$ the reverse edge zy has zero capacity and $\tau_{yz} = -\tau_{zy}$. Note that reverse edges (the zero-capacity edges in E) have zero or negative transit time. For a path P we use $\tau(P)$ to denote

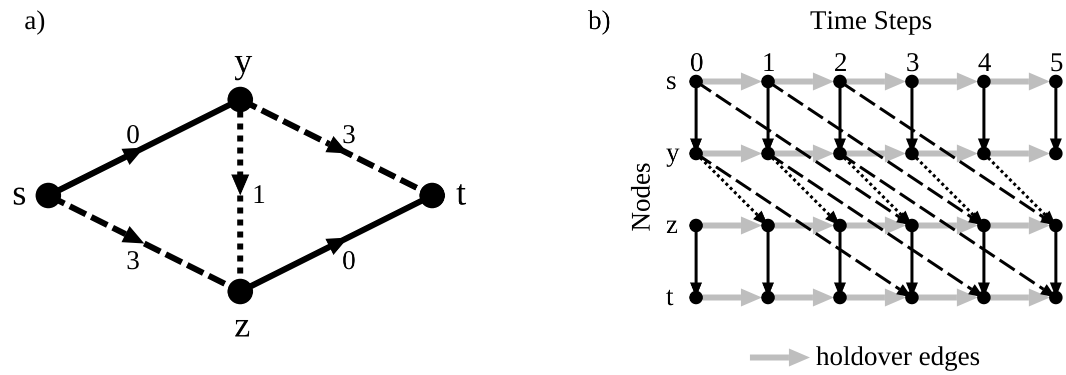

Figure 1: Dynamic Network with Transit Times & Time-Expanded Graph

the length (or total transit time) of the path P. Similar notation will also be used for cycles. We will also assume (again for simplicity only) that the only zero length cycles in E consist of pairs of opposite edges. We use k to denote $|S|$; likewise, $n = |V|$ and $m = |E|$. The maximum capacity is denoted by U and the maximum transit time by $\mathcal{T}$.

We refer to flows and circulations in G in the traditional sense as *static* flows and circulations. A *static* (s,t)*-flow* is a function f 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 for every node $y \in V$. A static flow or circulation f is *feasible* if it also satisfies the capacity constraints $f_{yz} \leq u_{yz}$ for every edge yz. The *residual network* of the static flow or circulation f is defined as $\mathcal{N}_f = (G, u^f, \tau, S)$, where the *residual capacity* function is $u^f_{yz} = u_{yz} - f_{yz}$. Note that the residual network of a flow will not satisfy some of our assumptions above: edges with positive capacity may have negative transit time, sources may have leaving edges, and sinks may have entering edges. The *value* of a static (s,t)-flow f is $|f| = \sum_y f_{sy}$. Multi-source multi-sink static flows are defined analogously; the value of flow f at a particular terminal x is denoted $|f|_x = \sum_y f_{xy}$.

A *finite horizon dynamic flow* f with time bound T is a static flow in the *time-expanded graph* $G(T) = (V(T), E(T))$. Each vertex $y \in V$ 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 θ 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 vertex y and time $\theta = 0, \ldots, T-1$. Dynamic flow f with time horizon T is a static flow in $G(T)$.

Example: Figure 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 1(b) shows the time-expanded graph $G(5)$ associated with this dynamic network. Every edge again has unit capacity, except for the infinite-capacity holdover edges.

An *infinite horizon dynamic flow* is a static flow in the infinite time-expanded graph $G(*)$, defined analogously to $G(T)$. We will view a finite horizon flow also as an infinite horizon dynamic flow by assuming that f is identically zero before time 0 and after time T. The *dynamic value* of f is the net flow out of the source s for all time steps, denoted by $|f| = \sum_\theta |f|_{s(\theta)}$. Multi-source multi-sink dynamic flows are defined analogously; in this case, $|f|_x = \sum_\theta |f|_{x(\theta)}$ for a particular terminal $x \in S$, denotes the net flow out of terminals $x(\theta)$ for a all times θ; note that for a sink x this is a non-positive amount. A dynamic flow is *feasible* if it satisfies the capacity constraints, *i.e.*, $f_{y(\theta)z(\theta+\tau_{yz})} \leq u_{yz}$, and the flow is balanced at all non-terminals, i.e., $|f|_{x(\theta)} = 0$ for all time θ and all $x \notin S$.

The following dynamic network flow problems apply to networks with one source and one sink. In the *maximum dynamic flow problem*, we are given a time bound T; we seek to maximize the value $|f|$ of a feasible dynamic flow f with time horizon T. In the *quickest flow problem*, we are given a flow amount v; we seek to find the minimum time T so that there is a feasible dynamic flow f with time horizon T and value $|f| = v$.

The *dynamic transshipment problem* $(\mathcal{N}, v, T)$ is a multiple-source and multiple-sink version of the maximum dynamic flow problem. We are given a dynamic network $\mathcal{N}$, a time bound T, and supplies v_x, where $v_x \geq 0$ for every source $x \in S^+$, $v_x \leq 0$ for every sink $x \in S^-$, and supply equals demand, *i.e.*, $\sum_{x \in S} v_x = 0$. The problem is to find a feasible dynamic flow f with time horizon T and $|f|_x = v_x$ for every terminal x, if such a flow exists. In the *quickest transshipment problem* $(\mathcal{N}, v)$, we are given supplies v_x for $x \in S$; the problem is to minimize the time T so that the resulting dynamic transshipment problem $(\mathcal{N}, v, T)$ is feasible. The *evacuation problem* is the special case of the quickest transshipment problem with a single sink.

3 Chain-Decomposable Flows

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

Ford and Fulkerson [14] 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. Temporally repeated flows, also known as *standard chain-decomposable flows*, are a simple subclass of chain-decomposable flows. All the algorithms of this paper compute chain-decomposable flows.

3.1 Temporally Repeated Flows

A *chain flow* $\gamma = \langle w, P \rangle$ is a static flow of value $w \geq 0$ along path (or cycle) P in a network $\mathcal{N} = (G, u, \tau, S)$. We denote the length of the chain flow by $\tau(\gamma)$; it is equal to $\tau(P)$, the total length of path P. Given time horizon T no less than $\tau(\gamma)$, any chain flow γ *induces* a dynamic flow by sending w units of flow along path P every time step from time zero until time $T - \tau(\gamma)$. The last w 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 multiset of proper chain flows. We say that Γ is a *chain decomposition* of static flow f if $\sum_{i=1}^{k} \gamma_i = f$. (In this sum, recall that each chain flow γ_i includes negative antisymmetric flow, so that flow on opposite edges automatically cancels.) We also say that Γ is a *standard chain decomposition* of f if all chain flows in Γ use edges in the same direction as f does. If every chain flow in Γ is no longer than time bound T, then Γ *induces* a dynamic flow by summing on each edge the dynamic flows induced by $\gamma_1, \ldots, \gamma_k$. We denote the resulting dynamic flow by $[\Gamma]^T$. If f was feasible in the static network G, then the resulting dynamic flow $[\Gamma]^T$ is feasible in $G(T)$. A dynamic flow computed in this manner is called a *temporally repeated flow*, and we denote it by $[\Gamma]^T$. In the absence of any

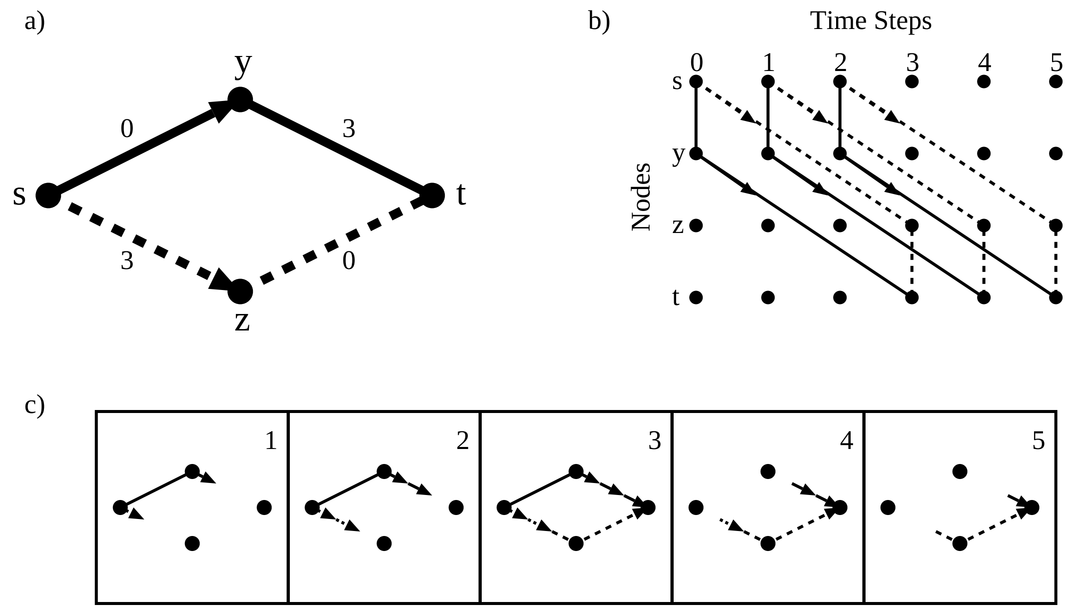

Figure 2: Temporally Repeated Flow

time horizon, Γ induces an infinite-horizon temporally repeated flow $[\Gamma]$ by repeating each chain flow endlessly starting at time zero.

Example: Consider the network of Figure 1(a) with time horizon $T = 5$. Figure 2(a) shows a maximum static flow in this network, decomposed into two chain flows. Figures 2(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.

Maximum Dynamic Flows. Temporally repeated flows are a very simple class of dynamic flows. Interesting dynamic flow problems cannot necessarily be solved by temporally repeated flows. Ford and Fulkerson [14] showed, however, that temporally repeated flows do suffice to solve the maximum dynamic flow problem. We briefly review their results here.

Consider a standard chain decomposition Γ 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 Γ. The dynamic value can be expressed as

$$|[\Gamma]^T| = \sum_{\gamma\in\Gamma} (T - \tau(\gamma) + 1)\,|\gamma| \tag{1}$$

$$= (T+1)|f| - \sum_{yz\in E^+} \tau_{yz} f_{yz}. \tag{2}$$

In the first equation, note that each chain flow γ in Γ contributes $|\gamma|$ units of flow for each time step from time 0 until time $T - \tau(\gamma)$. The second equation rearranges the terms of the first to show how this sum depends only on f and not on Γ. This implies that finding a maximum temporally repeated dynamic flow is equivalent to a minimum-cost circulation problem: assign every edge yz cost $c_{yz} = \tau_{yz}$ and introduce a return arc ts with infinite capacity and cost $-(T+1)$. Notice that a minimum-cost circulation in this network maximizes the formula above. Ford and Fulkerson also showed that there is always a maximum dynamic flow in the class of temporally repeated flows; thus a maximum dynamic flow can be computed via one minimum-cost circulation computation.

Extended Networks for Dynamic Flows with Many Terminals. 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 ψ to the network, connected to each source s_i by an infinite-capacity zero-transit-time edge ψs_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 $\mathcal{N}'$. Note that an extended network stretches our network assumptions slightly, since the transit times of arcs entering ψ are negative. When discussing an extended network $\mathcal{N}'$, 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 $E^+ \setminus \psi$).

3.2 Chain-Decomposable Flows

Unfortunately, the maximum dynamic flow problem is about the only one to yield to a temporally repeated solution. Most other dynamic flow problems are typically solved with time-expanded graphs. Here we introduce chain-decomposable flows. Chain-decomposable flows extend the notion of temporally repeated 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 backwards edge with negative transit time. Let $yz \in E^+$ be an edge with positive capacity. By assumption, $\tau_{yz} \geq 0$. The opposite edge zy has a non-positive transit time $-\tau_{yz}$. Such edges come up as backwards edges in residual graphs. Sending flow on a backwards edge zy undoes our previous decision to send flow on the edge $yz \in E+$. A unit of flow sent from y at time θ reaches z at time $\theta + \tau_{yz}$. To cancel this flow we send a unit of flow on the residual edge zy that starts from z at time $\theta + \tau_{yz}$ and arrives at y at time θ, *i.e.*, the flow on the residual edge travels backwards in time. This backward time travel is equivalent to sending one negative unit of flow from y at time θ; it reaches z at time $\theta + \tau_{yz}$. Figures 3(b1-b3) show the equivalence of positive flow moving backward in time and negative flow moving forward in time.

Example: Consider the network of Figure 1(a) with time horizon $T = 5$. Figure 4(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 3(a1-b3). Added together, this non-standard chain decomposition induces the dynamic flow of Figures 4(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; let γ_1 denote the solid-edged chain flow and γ_2 the dashed chain flow. Chain flow γ_2 uses opposite edge zy to cancel the flow on edge yz induced by γ_1. If the transit times of the network were different

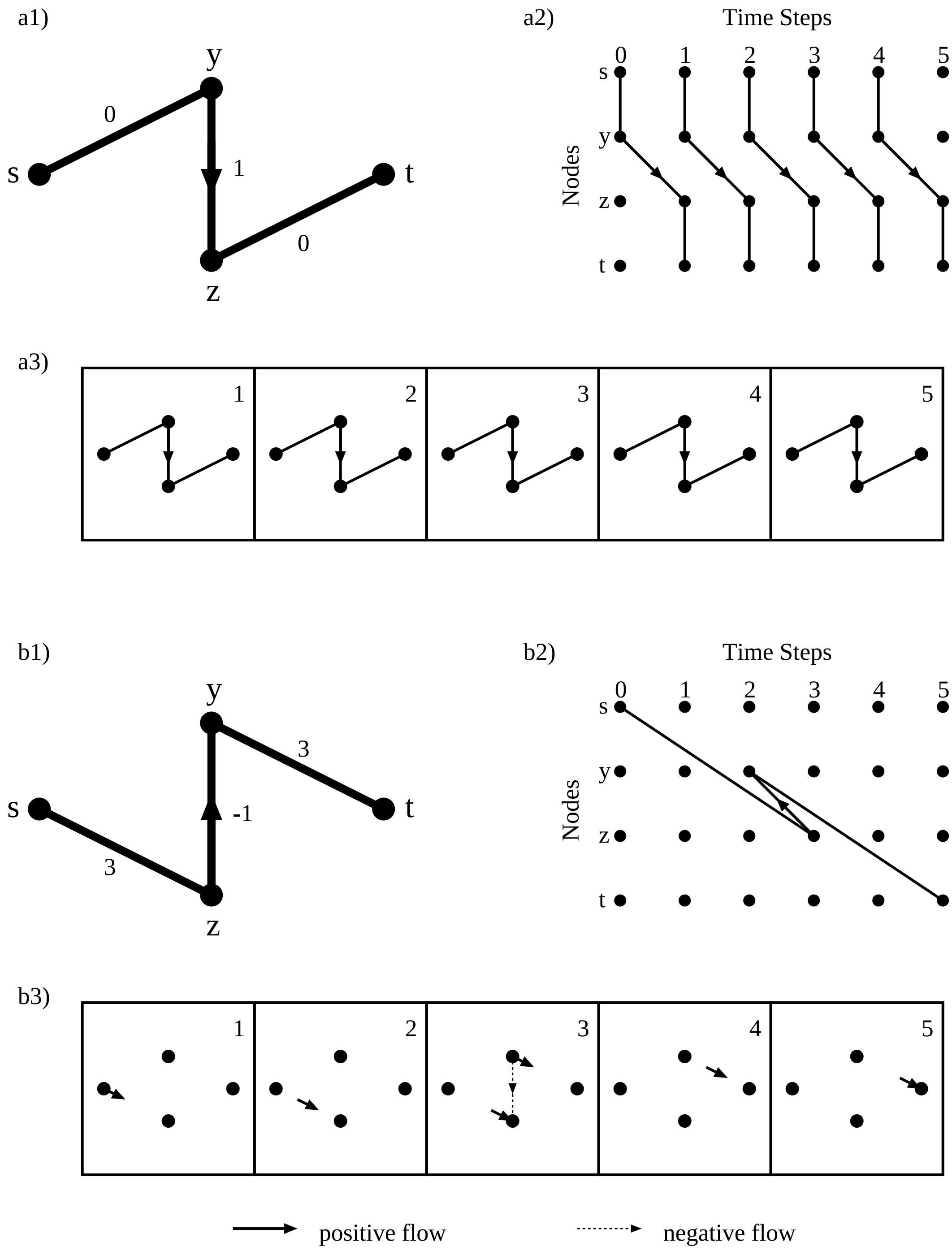

Figure 3: Chain Flows and Induced Dynamic Flows

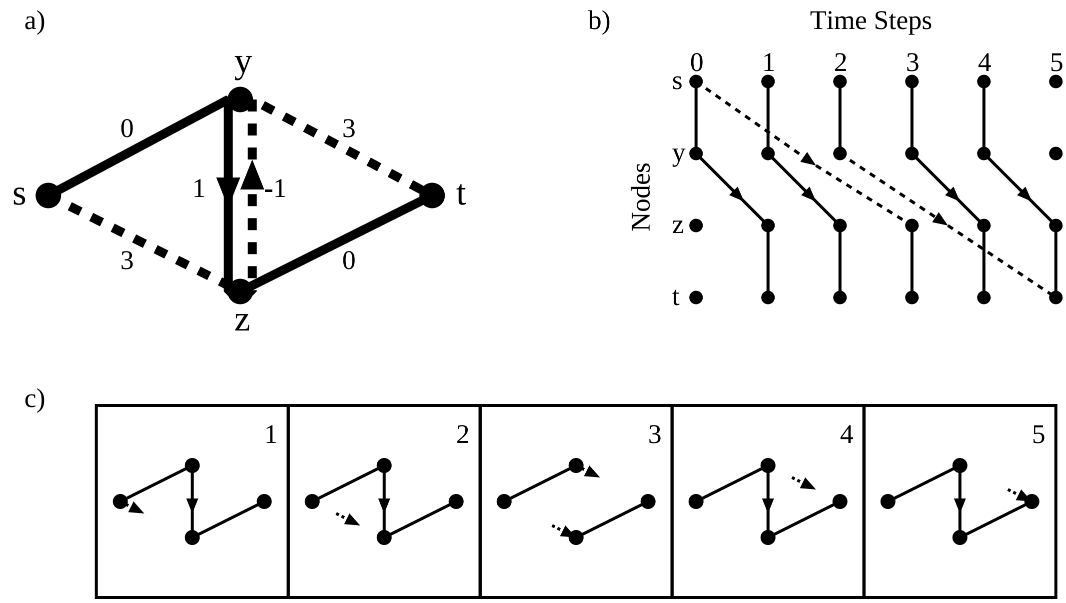

Figure 4: Non-Standard Chain-Decomposable Flow

so that γ_2 reached y before γ_1, however, then the flow induced by γ_2 would start cancelling on yz before γ_1 provided any flow there to cancel. The resulting dynamic flow would be infeasible, even though static flow $\gamma_1 + \gamma_2$ is feasible. Thus, in contrast to the trivial feasibility proofs of temporally repeated flows, we have to argue carefully to show that our chain-decomposable flows do not violate capacity constraints.

Zero Flow Chain Decompositions. We induce dynamic flow with non-standard chain decompositions of the zero flow in the extended network $\mathcal{N}'$. If Γ is a standard chain decomposition of the zero flow, then the induced dynamic flow $[\Gamma]$ is trivially zero. If we allow non-standard chain decompositions, however, then $[\Gamma]$ can be non-zero for a limited number of time steps; eventually, however, $[\Gamma]$ will be zero.

We use a particular kind of chain decomposition of the zero flow to induce dynamic flow. Each chain flow in Γ starts and ends in S. Such chains can also be viewed as cycles through ψ. This is our most general framework for chain-decomposable flows; every problem solved by temporally repeated 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 $\mathcal{N}'$ with $T = 1$. Figure 5 depicts this network with two different chain flows. Chain flow γ_1 pushes one unit of flow along cycle $(\psi s, st, t\psi)$; chain flow γ_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 temporally repeated flow $[\{\gamma\}]^T$, where γ is a unit-value chain flow on path (st).

The Value of a Chain-Decomposable Flow. We conclude this section by deriving the dynamic value of $[\Gamma]$ separately at each source. Our derivation shows a close relationship between dynamic flow value and static flow cost. Let the *extension* of Γ be the chain flows in Γ extended into cycles

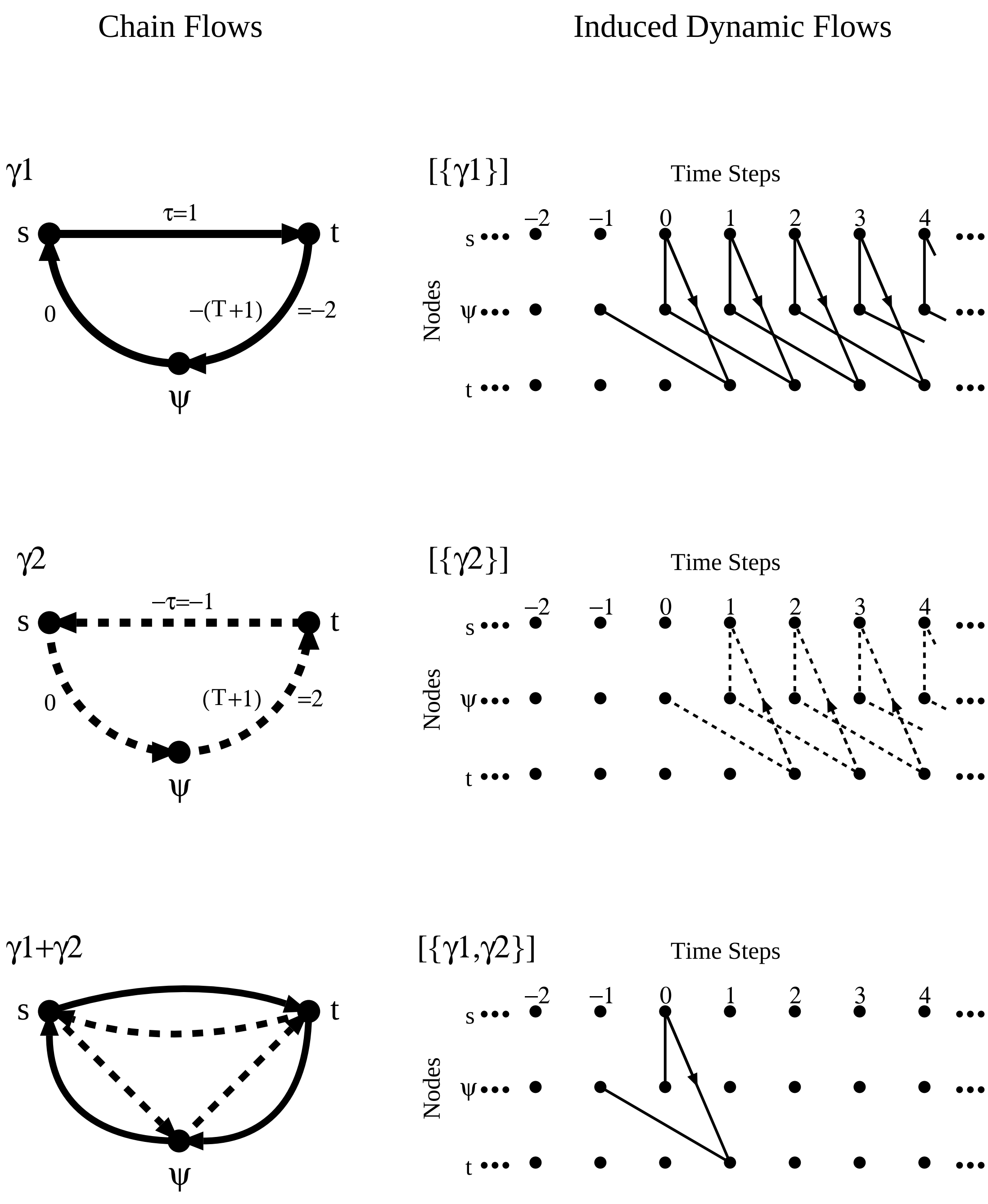

Figure 5: Zero-Flow Chain Decomposition

through ψ in the extended network. Let Γ_x be the multiset of all chain flows in Γ whose extensions use edge ψx (i.e., chain flows whose path starts at the terminal x), and let Γ'_x be the multiset of all chain flows in Γ whose extensions use opposite edge $x\psi$ (i.e., paths that end at terminal x). The assumption that Γ is a chain decomposition of the zero-flow implies that the total values of the chain flows in Γ_x and Γ'_x are the same (as the flows on the edge ψx and $x\psi$ must exactly cancel each other in the static flow). Flows along the chains Γ_x start from x at time 0, but the flow from any chain $\gamma' \in \Gamma'_x$ takes time $\tau(\gamma')$ to reach terminal x. Recall that chains ending at a source x cancel flow leaving from x. Consider a sufficiently large number θ of time steps, when $\theta \geq \tau(\gamma')$ for each chain $\gamma' \in \Gamma'_x$ and hence flow along each chain has reached the terminal x. At this point, every chain flow in Γ_x and Γ'_x has reached terminal x, and the net value of flow that has left x is

$$|[\Gamma]|_x \quad = \quad \sum_{\gamma \in \Gamma_x} \theta|\gamma| - \sum_{\gamma' \in \Gamma'_x} \left(\theta - \tau(\gamma')\right)|\gamma'| \;=\; \sum_{yz \in E^+} \left(\tau_{yz} \sum_{\gamma' \in \Gamma'_x} \gamma'_{yz} \right). \tag{3}$$

In the case of a single-source single-sink network (*e.g.*, an instance of the maximum dynamic flow problem), the above calculation is equivalent to Ford and Fulkerson's derivation of the value of a temporally repeated flow. Here we see again that the dynamic flow value is independent of the exact chain decomposition; the value depends only on the total cost of chain flows in Γ'_x — those which cancel flow on ψx. Notice that our formula implicitly includes the artificial edges of the extended network in E^+, whereas Ford and Fulkerson add that cost explicitly in their formula.

Lemma 3.1 *Suppose Γ is a proper chain decomposition of the zero flow, x is a source, and Γ'_x is defined as above. Then $|[\Gamma]|_x$ equals the total cost of all chain flows in the extension of Γ'_x.*

4 Lexicographically Maximum Dynamic Flows

In the *lexicographically maximum dynamic flow problem*, we are given an ordering of the k sources and sinks S as $s_1, \ldots, s_k$, and we seek a feasible dynamic flow with specified time horizon T that lexicographically maximizes the amounts leaving the terminals in the order $s_1, \ldots, s_k$. (This also lexicographically minimizes the amount of flow entering the sinks in the given order.) In this section, we show how to solve this problem. We then solve the quickest transshipment problem in Sections 5, 6, and 7 by reducing it to the problem considered in this section.

Minieka [30] and Megiddo [27] showed that in a static network with terminal set $S = \{s_1, \ldots, s_k\}$ there exists a feasible flow such that the amount of flow leaving the sets of terminals $S_i = \{s_1, \ldots, s_i\}$ is simultaneously maximum for each $i = 0, 1, \ldots, k$. Such a flow is clearly lexicographically maximum. This observation applies to the formulation of the lex max dynamic flow problem using the time-expanded graph $G(T)$, with $s_i(0)$ corresponding to source s_i and $s_j(T)$ corresponding to sink s_j. Note that the holdover edges allow flow to leave a source later than time zero and enter a sink earlier than time T.

The time-expanded graph is exponentially larger than the original dynamic network, so we cannot use Minieka's [30] or Megiddo's [27] algorithm directly. Our goal is to solve the problem in polynomial time using the original graph, and not the time-expanded graph. Beginning with the zero flow, our lex max dynamic flow algorithm computes successive layers of minimum cost (static) flows in the residual graphs of previous layers. See Figure 6. Each layer yields a standard chain decomposition Δ^i that is added to set Γ^i. At the end of the algorithm, $[\Gamma]$ is a lex max dynamic flow. Note that we use an infinite time horizon flow $[\Gamma]$, while our goal was to send the maximum

$V \leftarrow V \cup \{\psi\}$
$E^{k+1} \leftarrow E \cup \{\psi s_j : s_j \in S^+\}$ where $u_{\psi s_j} = \infty$ and $\tau_{\psi s_j} = 0$
$g^{k+1} \leftarrow 0$ (the zero flow)
$\Gamma^{k+1} \leftarrow \emptyset$
for $i \leftarrow k, \ldots, 1$ {
    $E^i = E^{i+1}$.
    if s_i sink, then add edge $s_i\psi$ to E^i with $u_{s_i\psi} = \infty$, and $\tau_{s_i\psi} = -(T+1)$.
        $f^i \leftarrow$ minimum-cost circulation in $\mathcal{N}^i_{g^{i+1}}$ using τ as edge costs
    if s_i source, then delete edge ψs_i from E^i
        $f^i \leftarrow$ minimum-cost maximum (ψ, s_i)-flow in $\mathcal{N}^i_{g^{i+1}}$ using τ as edge costs
    $g^i \leftarrow g^{i+1} + f^i$
    $\Delta^i \leftarrow$ standard chain decomposition of f^i
    $\Gamma^i \leftarrow \Gamma^{i+1} \cup \Delta^i$
}
Return $\Gamma = \Gamma^1$

Figure 6: Lexicographically Maximum Dynamic Flow Algorithm

possible amount in the finite time horizon T. In Theorem 4.7, we will show that the infinite horizon flow created by the algorithm is identically zero after time T.

We introduce a supersource ψ, connected to the network by infinite-capacity zero-transit-time artificial arcs ψs_i for all sources $s_i \in S^+$. Let $\mathcal{N}^{k+1}$ denote the resulting network, and g^{k+1} the zero flow in this network. Notice that g^{k+1} is a minimum cost circulation in $\mathcal{N}^{k+1}$ using transit times as costs, since there are no negative-cost positive-capacity edges in $\mathcal{N}^{k+1}$.

The algorithm consists of k iterations indexed in descending fashion. In iteration $i = k, \ldots, 1$ we consider terminal s_i. If s_i is a sink, we add an infinite capacity arc $s_i\psi$ to $\mathcal{N}^{i+1}$ with transit time $-(T+1)$ to create network $\mathcal{N}^i$, and compute a minimum cost circulation f^i in the residual network of the flow g^{i+1} in $\mathcal{N}^i$. We add f^i to g^{i+1} to obtain flow g^i. If s_i is a source, then we delete the edge ψs_i from $\mathcal{N}^{i+1}$ to create network $\mathcal{N}^i$, and compute a minimum-cost maximum flow f^i from ψ to s_i in the residual network of the flow g^{i+1} in $\mathcal{N}^i$. We add this minimum-cost flow f^i to the flow g^{i+1} to obtain flow g^i. Note that some chain flows from ψ to source s_i will use arc ψx for some sink x. These chain flows leave x at time $T+1$ and cancel flow that would have arrived at x too late. Each flow f^i yields a standard chain decomposition Δ^i. The assumption that there are no zero length cycles in $\mathcal{N}$ guarantees that there are no cycles in the decomposition. The chain flows are accumulated into chain decomposition $\Gamma^i = \Gamma^{i+1} \cup \Delta^i$.

We need the following definitions to analyze the algorithm. We say that chain flow $\gamma = \langle v, P\rangle$ *touches* an edge yz if $yz \in P$ or $zy \in P$, and that γ *covers* yz at time θ if $[\{\gamma\}]_{yz}(\theta) \neq 0$. Note that the infinite horizon of $[\Gamma]$ means that if γ covers yz at time θ, then γ also covers yz at all times after θ.

The artificial edges and the assumption that in the original graph sources have no entering edges in E^+ guarantee that after iteration i, the node s_i is balanced in the static flow g^i. The flow f^i used to create g^i was of minimum cost; this implies that the circulation g^i is of minimum cost.

Lemma 4.1 *For any iteration i, static flow g^i is a minimum-cost circulation in network $\mathcal{N}^i$.*

The feasibility of the resulting dynamic flow rests fundamentally on the following lemma. Let

$p^i(x)$ denote the minimum cost of a path from supersource ψ to vertex $x \in V$ in the residual network of the flow g^i in $\mathcal{N}^i$.

Lemma 4.2 *For any vertex $x \in V$ and any iteration i, $p^i(x) \geq p^{i+1}(x)$.*

Proof: Recall that the flow f^i is minimum cost in the residual network of g^{i+1} in $\mathcal{N}^i$. For this proof, let $q^i(x)$ denote the cost of the shortest path from ψ to x in this residual network. First note that $q^i(x) \geq p^{i+1}(x)$. This is because $\mathcal{N}^i$ is obtained from $\mathcal{N}^{i+1}$ by either deleting an edge, or by adding an edge that enters ψ. Neither of these changes can create a shorter path from ψ to x.

Next we will show that $p^i(x) \geq q^i(x)$. The cost of the shortest path $p^i(x)$ is the cost of sending one additional unit of flow from ψ to x in the residual network of the flow g^i. Let f denote such a unit flow. Recall that g^i was obtained from g^{i+1} by adding the minimum-cost flow f^i. Consider the sum of these two flows, $f^i + f$. The sum is a flow in the residual network of g^{i+1} in $\mathcal{N}^i$, and it can be decomposed into (1) a unit of flow from ψ to x, and (2) a flow satisfying the same demands as f^i. Let f' denote the unit flow, and f'' denote the remaining flow. The flow f^i is minimum cost, hence the cost of f'' is at least the cost of f^i. Therefore, the cost of f' is at most the cost of f. Now $p^i(x)$ is the cost of f, and $q^i(x)$ is at most the cost of f', so we see that $q^i(x) \leq p^i(x)$. $\Box$

Lemma 4.3 *Suppose $\gamma \in \Delta^{i-1}$ and $y, z \in V$. Then γ does not cover yz at any time before $p^i(y)$.*

Proof: By definition, the residual network $\mathcal{N}^i_{g^i}$ contains no (s, y)-path of length less than $p^i(y)$. The path of any chain flow in Δ^{i-1} is a path in $\mathcal{N}^i_{g^i}$, and hence cannot cover yz at any time before $p^i(y)$. $\Box$

Lemma 4.4 *Suppose $\gamma \in \Delta^i$ and $y, z \in V$. If γ touches yz, then γ covers yz at time $p^i(y)$.*

Proof: By contradiction. If γ touches yz, but does not cover it at time $p^i(y)$, then the part of the path from γ from s to y is longer than $p^i(y)$. This means that this part of the path of γ gives a residual path from y to s in $\mathcal{N}^i_{g^i}$ of cost less than $-p^i(y)$. Together with the definition of $p^i(y)$ this means that the residual network $\mathcal{N}^i_{g^i}$ has a negative cycle, contradicting Lemma 4.1. $\Box$

Theorem 4.5 $[\Gamma]$ *is a feasible dynamic flow.*

Proof: We prove that $[\Gamma^i]$ obeys all capacity constraints by reverse induction on i. Γ^k is a standard chain decomposition of g^k, therefore $[\Gamma^k]$ obeys all capacity constraints. Suppose $1 \leq i \leq k$, and $[\Gamma^i]$ is feasible; we show that $[\Gamma^{i-1}]$ is feasible. Recall that $\Gamma^{i-1} = \Gamma^i \cup \Delta^{i-1}$, and therefore $[\Gamma^{i-1}] = [\Gamma^i] + [\Delta^{i-1}]$, where we add the two dynamic flows $[\Gamma^i]$ and $[\Delta^{i-1}]$ edge by edge.

Consider any edge yz. If $f^{i-1}(yz) = 0$, then $[\Gamma^{i-1}]_{yz}(\theta) = [\Gamma^i]_{yz}(\theta)$ for all times θ, and so by the induction hypothesis the capacity constraint is not violated. Next assume that $f^{i-1}(yz) > 0$. (The other direction is considered as opposite edge $z(\theta')y(\theta)$.) Consider first a time $\theta \leq p^i(y)$. By Lemma 4.3 no chain flow in Δ^{i-1} covers yz at any time before $p^i(y)$, and hence we have that $[\Gamma^{i-1}]_{yz}(\theta) = [\Gamma^i]_{yz}(\theta)$ as before. For any time step $\theta \geq p^{i-1}(y)$ Lemma 4.4 implies that all chain flows in Δ^{i-1} that touch yz cover yz at time $p^{i-1}(y)$. This implies that $[\Delta^{i-1}]_{yz}(\theta) = f^{i-1}(yz)$, hence $[\Gamma^{i-1}]_{yz}(\theta) = g^{i-1}_{yz}$, and so the capacity constraint is not violated. Finally, consider a time θ such that $p^i(y) \leq \theta \leq p^{i-1}(y)$. Using the above argument for i instead of $i - 1$, we see that $[\Gamma^i]_{yz}(\theta) = g^i_{yz}$. At time θ, a subset of the chains in Δ^{i-1} may be covering yz. Recall that $f^{i-1}(yz) > 0$, and Δ^{i-1} is a standard chain decomposition of f^{i-1}. This implies that

$$[\Gamma^i]_{yz}(\theta) \leq [\Gamma^{i-1}]_{yz}(\theta) \leq [\Gamma^i]_{yz}(\theta) + f^{i-1}(yz).$$

We have that $g^i(yz) = [\Gamma^i]_{yz}(\theta)$, and $[\Gamma^i]_{yz}(\theta) + f^{i-1}(yz) = g^{i-1}(yz)$. Hence we get that $g^i(yz) \leq [\Gamma^{i-1}]_{yz}(\theta) \leq g^{i-1}(yz)$, and so the capacity constraint is not violated.

Finally, we consider flow conservation. These constraints are trivial except at the source nodes: the chain decomposition Γ includes chains flows terminating at sources. However, the validity of the capacity constraints and the assumption that in the original graph G no source has incoming edges, guarantee that no source ever has net incoming dynamic flow in G. □

Theorem 4.6 $[\Gamma]$ *is a lexicographically maximum dynamic flow.*

Proof: Our proof relies on the infinite time-expanded graph $G(*)$. (Note, however, that Theorem 4.7 shows that the flow ends at time T). Given any index $i : 0 \leq i \leq k$, we construct a finite-capacity cut C_i in the time-expanded graph, and show that $[\Gamma]$ saturates C_i. Cut C_i separates the source set $\{s_j(0) : j \leq i; s_j \in S^+\} \cup \{s_j(T) : j \leq i; s_j \in S^-\}$ from the sink set $\{s_j(T) : j > i; s_j \in S^-\} \cup \{s_j(0) : j > i; s_j \in S^+\}$. This saturated cut implies that the amount of dynamic flow leaving the set of terminals $S_i = \{s_1, \ldots, s_i\}$ is maximum.

The cut is defined as $C_i = \{x(\theta) : x \in V \text{ and } \theta \geq p^{i+1}(x)\} \cup \{s_j(T) : j \leq i, s_j \in S^-\}$. It is easy to see that C_i contains the source set, because the artificial edges in network Γ^{i+1} guarantee that $p^{i+1}(s_j) = 0$ for all sources s_j such that $j \leq i$. It is not much harder to see that C_i contains no element of the sink set by considering each iteration $j \geq i+1$. If terminal s_j is in S^-, then iteration j adds infinite-capacity artificial edge (s_j, s) 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 (s, s_j)-path so that $p^j(s_j) = \infty$. These facts together with Lemma 4.2 imply that $p^{i+1}(s_j) > T$ 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 $[\Gamma]$ saturates cut C_i. We compute the capacity of the cut C_i as follows. 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 $\theta < p^{i+1}(y)$, otherwise we distinguish non-holdover edges from holdover edges. (1) If $\theta < p^{i+1}(y)$ then the definition of C_i implies that $y = s_j$, s_j is a sink, and $j \leq i$. Recall our assumption that each sink in S^- has no outgoing capacity in the input network G; 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 θ such that $p^{i+1}(y) \leq \theta < p^{i+1}(z) - \tau_{yz}$. This analysis applies to all original edges (i.e., not including the artificial edges adjacent to the superterminal ψ). Recall that $E^+ \setminus \psi$ denotes this set of edges. Thus, the total capacity of cut C_i is

$$\sum_{yz \in E^+ \setminus \psi} \max\{0, p^{i+1}(z) - \tau_{yz} - p^{i+1}(y)\} u_{yz}.$$

By the definition of the p^{i+1} function, $p^{i+1}(z) = \tau_{yz} + p^{i+1}(y)$ whenever $0 < g^{i+1}_{yz} < u_{yz}$ and $p^{i+1}(z) \leq \tau_{yz} + p^{i+1}(y)$ whenever $g^{i+1}_{yz} < u_{yz}$. Therefore, the above formula reduces to

$$\sum_{yz \in E^+ \setminus \psi} \left(p^{i+1}(z) - \tau_{yz} - p^{i+1}(y)\right) g_{yz} = \sum_{yz \in E^+ \setminus \psi} \left(p^{i+1}(z) - p^{i+1}(y)\right) g^{i+1}_{yz} - \sum_{yz \in E^+ \setminus \psi} \tau_{yz} g^{i+1}_{yz}.$$

The first term of the latter sum is equal to

$$\sum_y p^{i+1}(y) \left(\sum_{z : zy \in E^+ \setminus \psi} g^{i+1}_{zy} - \sum_{z : yz \in E^+ \setminus \psi} g^{i+1}_{yz} \right).$$

Flow conservation implies that this sum cancels itself out except at terminal nodes. What is left at a terminal node s_j is $p^{i+1}(s_j)$ times the net flow g^{i+1} into s_j. Notice that for any source s_j in S^+, if $g^{i+1}_{s_j z} > 0$ then $p^{i+1}(s_j) = 0$; and for any sink s_j in S^-, if $g^{i+1}_{ys_j} > 0$ then $p^{i+1}(s_j) = T + 1$. Thus, the first term is equal to $\sum_{x \in S^-} \tau_{xs} g^{i+1}_{xs} - \sum_{x \in S^+} \tau_{sx} g^{i+1}_{sx}$, and so we can simplify the remaining terms by considering all the artificial edges of network Γ^{i+1} and deriving the capacity of cut C_i to be $-\sum_{yz \in E^+} \tau_{yz} g^{i+1}_{yz}$.

The value of $[\Gamma]$ across C_i is equal to the net flow out of S_i, derived as follows: Observe that g^{i+1} is the sum of all chain flows into $S \setminus S_i$. Therefore, Lemma 3.1 implies that the total cost of g^{i+1} 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 $[\Gamma]$ across it. □

Dynamic flow $[\Gamma]$ appears to be an infinite horizon dynamic flow, and so in Theorem 4.6 we explicitly ignore any flow in $[\Gamma]$ reaching some sink s_j after time T; however, $[\Gamma]$ is actually a finite horizon dynamic flow in G with time bound T.

Theorem 4.7 $[\Gamma]$ *has time horizon* T.

Proof: For any $y, z \in V$ and any time $\theta \geq T+1$, we claim that $[\Gamma]_{yz}(\theta) = 0$. Consider all the chain flows in Γ that touch yz. (If there are none, the claim is trivially true.) Notice that Lemma 4.1 and the assumption that G has no zero length cycles imply that the static flow g^1 is zero. If a chain flow γ touches yz then it covers yz after some time that depends on γ. This implies that $[\Gamma]_{yz}(\theta) = 0$ for any sufficiently large time θ, and so there is no flow left in the network after a sufficiently large time T'. Notice that there is no flow entering any a sink $x \in S^-$ after time T (because of the canceling effect of chain flows using arc sx). Assume by 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 there is no flow left in the network after time T'. □

Theorem 4.8 *A lexicographically maximum dynamic flow with* k *sources and sinks can be computed via* k *minimum cost flow computations.*

5 Testing Dynamic Transshipment Feasibility

In this section we describe two strongly polynomial algorithms: (1) testing the feasibility of a dynamic transshipment problem $(\mathcal{N}, v, T)$, and (2) finding the optimal time T for a quickest transshipment problem $(\mathcal{N}, v)$. The first algorithm is used as a subroutine in the second.

Let S be the set of terminals in network $\mathcal{N}$. For a subset A of S, let $v(A)$ denote the total supply of A (that is, $\sum_{x \in A} v_x$), and let $o(A)$ denote the maximum amount of flow that the sources in A can send to the sinks in $S \setminus A$ in time T without regard to the needs of other terminals. Notice that computing $o(A)$ is equivalent to a single-source single-sink maximum dynamic flow problem (a single source connected to the sources in A and a single sink connected to the sinks in $V \setminus A$). We will use $\text{MCF}(m, n)$ to denote the time required for a min-cost flow computation on a network with n nodes and m edges. From the above we get that $o(A)$ for a set A can be determined in $O(\text{ MCF}(m, n))$ time [14]. For a dynamic transshipment problem $(\mathcal{N}, v, T)$ to be feasible, we must have that $o(A) \geq v(A)$ for every set $A \subseteq S$. Klinz [26] observed that this condition is equivalent to the cut condition of feasibility on the exponentially large time-expanded graph, and therefore the condition is also sufficient. We call a subset of sources A *violated* if $o(A) < v(A)$.

Theorem 5.1 (Klinz [26]) *The dynamic transshipment problem* $(\mathcal{N}, v, T)$ *is feasible if and only if* $o(A) \geq v(A)$ *for every subset* $A \subseteq S$.

Next we show how to test feasibility in polynomial time. A function $o(.)$ is submodular if it satisfies $o(A) + o(B) \geq o(A \cap B) + o(A \cup B)$ for every pair of subsets A and B. It is not hard to show that the $o(.)$ function above is submodular (see Megiddo [27]).

To simplify running times for the rest of the paper, we assume $k\,\mathrm{MCF}(m, n)$ is at least as large as the time required for a $k \times k$ matrix inversion. Inverting a $k \times k$ matrix takes time $O(k^3)$, and much less if we use fast matrix multiplication. Note that $k \leq n \leq m$ for our problem. All min-cost flow computations known require at least $O(mn)$ time, so k min-cost flow computations would require time $\Omega(knm)$.

Theorem 5.2 *A violated set in a dynamic transshipment problem* $(\mathcal{N}, v, T)$ *can be found in strongly polynomial time, and also in* $O(2^k\,\mathrm{MCF}(m, n)\,)$ *time, or in* $O(k^2\,\mathrm{MCF}(m, n) \log(nTU))$ *time.*

Proof: The $O(2^k\,\mathrm{MCF}(m, n)\,)$ time is immediate. To obtain the other two versions, 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 by minimizing the submodular function b. A strongly polynomial algorithm for minimizing submodular functions was given by Grötschel, Lovász and Schrijver [17].

For the $O(k^2\,\mathrm{MCF}(m, n) \log(nTU))$ time version, we use Vaidya's algorithm [35]. Consider the polytope $\mathcal{P} \subset \mathbf{R}^{|S|}$ of all transshipment supplies and demands v'_x for $x \in S$, such that $(\mathcal{N}, v', T)$ is feasible. By Theorem 5.1 we have that

$$P = \{v' \in \mathbf{R}^{|S|} \text{ such that } v'(A) \leq o(A) \text{ for all subsets } A \text{ of } S\}.$$

Recall that the function $o(.)$ is submodular, and hence P is an extended polymatroid, and any objective function can be optimized over P via the greedy algorithm (see in [17]). Optimization over the polytope $\mathcal{P}$ can be done in $O(k\,\mathrm{MCF}(m, n)\,)$ time.

We apply Vaidya's algorithm to the polar polytope $\mathcal{P}^* = \{y : yx \leq 1 \ \forall x \in \mathcal{P}\}$. It is easy to see that $P^{**} = P$, and so a supply vector v is feasible, i.e., $v \in P$ if and only if $yv \leq 1$ for every $y \in \mathcal{P}^*$; therefore testing feasibility of v is equivalent to optimization over $\mathcal{P}^*$.

Vaidya's algorithm for optimization over $\mathcal{P}^*$ needs an oracle to test membership in $\mathcal{P}^*$. The polytope $\mathcal{P}$ is bounded; therefore $\mathcal{P}^*$ is full-dimensional. Vaidya's algorithm takes $O(kL)$ iterations on full-dimensional polytopes in k dimensions, where L in our case is $O(\log(nTU))$. Each iteration consists of the inversion of a $k \times k$ matrix and one call to the oracle. Notice that the running time of the matrix inversions are dominated by the k minimum-cost flow computations.

If v is infeasible, Vaidya's algorithm finds $y \in \mathcal{P}^*$ such that $yv > 1$. We claim that $A(\alpha) = \{i : y_i \geq \alpha\}$ is a violated set for some value of α. Assume for notational simplicity that $y_1 \geq y_2 \geq \ldots \geq y_k$ and let $y_{k+1} = 0$. The polytope $\mathcal{P}$ is an extended polymatroid, therefore y is maximized over $\mathcal{P}$ by the greedy algorithm, and the maximum is $\sum_i (y_i - y_{i-1}) o(A(y_i))$. By the assumption that $y \in \mathcal{P}^*$ this maximum is at most 1. We also have that $yv > 1$, and we can write yv as $\sum_i (y_i - y_{i-1}) v(A(y_i))$. Therefore we must have that $v(A(y_i)) > o(A(y_i))$ for some i. $\Box$

Finding the Optimal Time for Quickest Transshipment. To solve a quickest transshipment problem $(\mathcal{N}, v)$, we must find the minimum time T such that the dynamic transshipment problem $(\mathcal{N}, v, T)$ becomes feasible. This can be done in polynomial time using a binary search and Theorem 5.2.

To analyze the binary search, we need an upper bound for the optimal time T. Theorem 5.1 implies that there is some set $W \subset S$ 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 evacuation problem. 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. An upper bound for this time is $n\mathcal{T}$. Let V equal the sum of all positive supplies: $v(S^+)$. In a single-source single-sink setting, it is easy to see that after $n\mathcal{T}$ 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 for the quickest transshipment problem can be bounded: $T \leq n\mathcal{T} + V$.

We can also find the optimal time T in strongly polynomial time using Megiddo's parametric search [28] instead of the binary search. We briefly explain this technique in the following paragraph.

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 λ. 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 λ (unless $b_i = b_j$). All we need to know about λ, however, is its relationship to the critical value λ^*, determined by $(a_i + \lambda^* b_i) = (a_j + \lambda^* b_j)$. In our case, we have a network with one linear parameter $\lambda = T$. We can determine whether $T \leq T^*$ or $T \geq T^*$ by running the original algorithm $\mathcal{A}$ on the non-parameterized network that substitutes critical value T^* for parameter T. 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 [29] that uses parallel algorithms to solve parametric optimization problems efficiently.

Theorem 5.3 *The optimal time T for the quickest transshipment problem can be found in time $O(2^k \operatorname{MCF}(m,n) \log(nV\mathcal{T}))$, or in time $O(k^2 \operatorname{MCF}(m,n) \log^2(nUV\mathcal{T}))$, and also in strongly polynomial time.*

We will assume from now on that we know the optimal time bound T for any quickest transshipment problem $(\mathcal{N}, v)$. We then need to find a dynamic transshipment with time bound T.

6 Integral Dynamic Transshipment Algorithm

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 an algorithm to solve the lex max dynamic flow problem in Section 4.

Let S be the set of terminals in the original network $\mathcal{N}$. Our algorithm creates the new network in two phases by attaching a new set of terminals S' to S. In the first phase S' contains one source x_0 for each source $x \in S^+$, connected by edge x_0x with infinite capacity and zero transit time; likewise, S' contains one sink x_0 for each sink $x \in S^-$, connected by edge xx_0 with infinite capacity and zero transit time. Define $v' : S' \to \mathbb{N}$ based on $v : S \to \mathbb{N}$ in the obvious manner: $v'_{x_0} = v_x$. Clearly $(\mathcal{N}', v', T)$ is equivalent to $(\mathcal{N}, v, T)$. Later, we will add other terminals to S' and divide

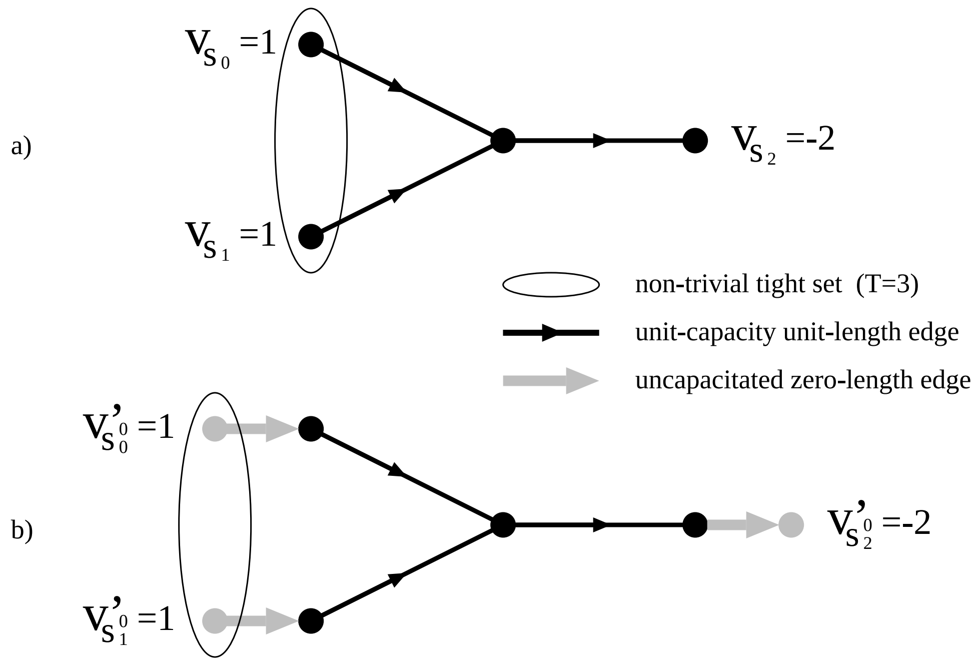

Figure 7: Transforming $(\mathcal{N}, v, T)$ to $(\mathcal{N}', v', T)$

the demand or supply of an original terminal x between the new terminals, $x_0, x_1, \ldots \in S'$ that will be attached to x. We will maintain the property that any solution to the new problem $(\mathcal{N}', v', T)$ yields a solution to the original problem $(\mathcal{N}, v, T)$. The algorithm will treat the terminals x_0 for all $x \in S$ that were added to S' in the first phase differently. We will call these *first terminals*. The goal is to arrange the new problem so that a lex max flow yields the desired dynamic transshipment.

The second phase of the algorithm is a loop. Each iteration of the loop adds new terminals to S'. Each new terminal x_i is attached to a terminal x 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 x_i based on these restrictions and adjusts the supply of the corresponding first terminal x_0 so that for any $x \in S$, the total supply of terminals $x_0, x_1, \ldots$ in S' associated with x is v_x. All terminals attached to sources will have positive supply, while all terminals attached to sinks will have negative supply, i.e., demand. This maintains the invariant that any solution to $(\mathcal{N}', v', T)$ yields a solution to $(\mathcal{N}, v, T)$.

We call a subset A of terminals *tight* if $o(A) = v(A)$. We call the sets $\emptyset$ and S *trivial* tight sets, all other tight sets are non-trivial. (Note that the set S is tight due to our assumption that $\sum_{x \in S} v_x = 0$.)

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. The following theorem 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:

Theorem 6.1 *Suppose $\mathcal{C}$ is a chain of tight subsets of S' in dynamic transshipment problem $(\mathcal{N}', v', T)$, and $|\mathcal{C}| = |S'| + 1$. Then the dynamic transshipment problem $(\mathcal{N}', v', T)$ can be solved by a single lex max dynamic flow computation in $\mathcal{N}'$.*

Proof: If $|\mathcal{C}| = |S'| + 1$, then there is an ordering of S' as $s_1, \ldots, s_{k'}$ such that $\{s_1, \ldots, s_i\}$ is a tight set in $\mathcal{C}$ for all $i = 1, \ldots, k'$. This means that any feasible flow satisfying all supplies must be lexicographically maximum with respect to the above order. Conversely, this also means that any solution to the lex max dynamic flow problem corresponding to this order yields a solution to the original dynamic transshipment problem. □

We measure the progress of the algorithm by the formula $|\mathcal{C}| - |S'|$. Initially, $\mathcal{C} = \{\emptyset, S'\}$, and S' contains one new terminal x_0 for every terminal $x \in S$ of the original problem. So $|\mathcal{C}| - |S'| = 2 - k$.

By the above theorem, we are done when $|\mathcal{C}| - |S'| = 1$. The goal of an iteration of the algorithm is to increase the value of this expression. Let $Q \subset R$ be adjacent sets in $\mathcal{C}$. (By adjacent we mean $\nexists A \in \mathcal{C} : \ Q \subset A \subset R$.) If all pairs of adjacent sets have $|R \setminus Q| = 1$ then the number of sets in $\mathcal{C}$ is $|S'| + 1$, and hence there is nothing to do. We will maintain the invariant that if $|R \setminus Q| > 1$ for two adjacent sets R and Q, then every terminal in $R \setminus Q$ is one of the first terminals x_0 in S'. Below, we describe how one iteration of the algorithm increases $|\mathcal{C}| - |S'|$ by one, maintains the above invariant, the feasibility of $(\mathcal{N}', v', T)$, and the invariant that any solution to $(\mathcal{N}', v', T)$ yields a solution to $(\mathcal{N}, v, T)$. Thus, the given dynamic transshipment problem is reduced to an equivalent lex max dynamic flow problem after $k - 1$ iterations.

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 towards this goal is to add more terminals to S'. Done arbitrarily, these terminals would take the algorithm farther from the goal; but new terminals are connected to the network through arcs with carefully computed capacities and transit times. By assigning supplies to new terminals according to these capacities and transit times (and adjusting the supplies of other terminals 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 $|R \setminus Q| > 1$. If no such Q and R exist, then $|\mathcal{C}| - |S'| = 1$ and we are done. Recall the invariant that $R \setminus Q$ consists of first terminals.

Let s_0 be a source in $R \setminus Q$. (The treatment of sinks is symmetric.) We first check if the set $Q \cup \{s_0\}$ is tight; if so, then we can add it to chain $\mathcal{C}$ and the current iteration is done. If the set $Q \cup \{s_0\}$ is not tight, however, then let s_0 be adjacent to s, one of the original sources in S; suppose there are $i - 1$ other sources in S' adjacent to s. We will add two new sources s_i and s_{i+1} to S' in this iteration. The edge $s_i s$ will have zero transit time and finite capacity α, while the edge $s_{i+1} s$ will have unit capacity and finite transit time δ. The goal of the following steps is to determine the values for α and δ.

Consider new source s_i, added to S' and connected to s by a parameterized α-capacity zero-length arc. Let $Q' = Q \cup \{s_i\}$. Recall that $o(A)$ is the maximum dynamic flow out of a subset of terminals A. Now this amount depends on the choice of the capacity α. We will use $o^\alpha(A)$ to denote the maximum dynamic flow out of a subset of terminals A in the parametric network. Define new supply v^α: the supply of s_i is $v^\alpha_{s_i} = o^\alpha(Q') - o^\alpha(Q)$; the supply of s_0 is reduced by $v^\alpha_{s_i}$; and other terminal supplies remain unchanged. Note that this problem may be infeasible, for example, if the supply of s_0 is reduced to be negative.

Consider the above parameterized dynamic transshipment problem $(\mathcal{N}^\alpha, v^\alpha, T)$. We will look for the maximum value of α that defines a feasible problem. Notice that if $\alpha = 0$, then the maximum flow out of s_i is zero: the problem is equivalent to $(\mathcal{N}', v', T)$ and is therefore feasible. At the opposite extreme, if $\alpha = \infty$ then we claim that the problem $(\mathcal{N}^\alpha, v^\alpha, T)$ is infeasible. In this case, the new terminal s_i is connected to the network in exactly the same way as s_0, and hence the supply of the new terminal is $o(Q \cup \{s_0\}) - o(Q)$. Recall that the set $Q \cup \{s_0\}$ is not tight, but

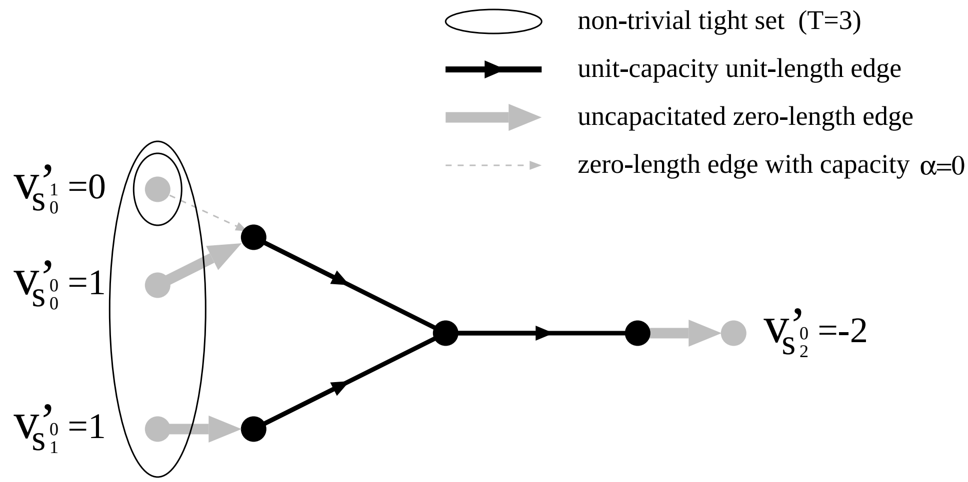

Figure 8: Connecting New Source s_0^1 with Parameterized Capacity

the set Q is tight. Thus, reducing the supply of s_0 by this amount results in a negative supply at s_0, which has no incoming capacity, and is therefore infeasible. Thus, we can binary search for an integer α^* 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 original source $s \in S$.

Figure 8 shows the result of choosing source s_0^0 in the network of Figure 7(b) and completing the α-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.

Next we will add source s_{i+1} to the dynamic transshipment problem $(\mathcal{N}^{\alpha^*}, v^{\alpha^*}, T)$. Connect s_{i+1} to S', by a parameterized δ-length unit-capacity arc. Now $o(A)$, the maximum dynamic flow out of a subset of terminals A, depends on the choice of the length δ. We will use $\overline{o}^\delta(A)$ to denote the maximum flow in this parameterized network. Let $Q'' = Q' \cup \{s_{i+1}\}$. Define new supply $\overline{v}^\delta$: the supply of s_{i+1} is $\overline{v}^\delta_{s_{i+1}} = \overline{o}^\delta(Q'') - \overline{o}^\delta(Q')$; the supply of s_0 is reduced by $\overline{v}^\delta_{s_{i+1}}$; and other terminal supplies remain unchanged.

Our goal will be to find the minimum value of δ for which the new parameterized dynamic transshipment problem $(\overline{\mathcal{N}}^\delta, \overline{v}^\delta, T)$ is feasible. Notice that if $\delta = T+1$, then the maximum flow out of s_{i+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 δ^* such that $\delta = \delta^*$ yields a feasible problem but $\delta = \delta^* - 1$ yields an infeasible problem.

Figure 9 shows the completion of a one-iteration run of the algorithm. (The rest of the run is illustrated in Figures 7 and 8.) The δ-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 6.1.

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

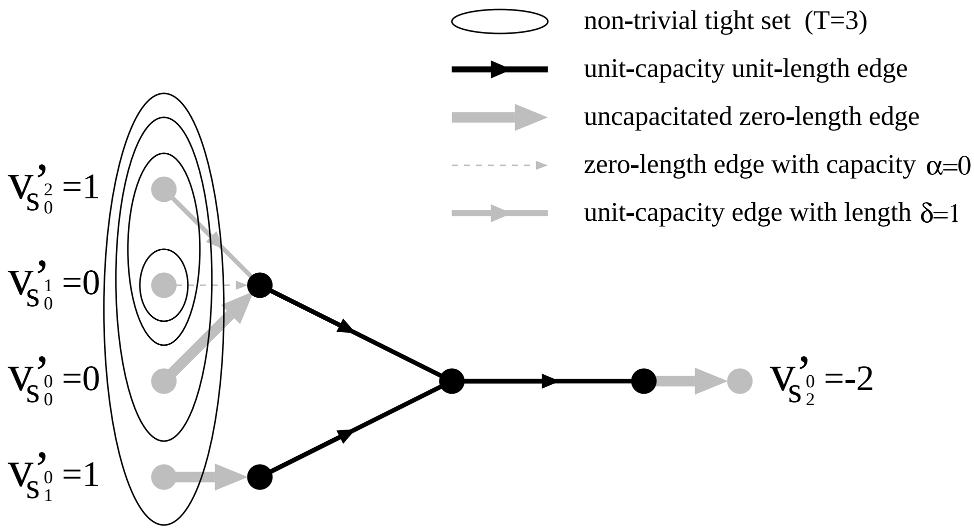

Figure 9: Connecting New Source s_0^2 with Parameterized Transit Time

depends on the following properties of the new dynamic transshipment problem $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$:

Property 6.2 *Q' and Q'' are both tight sets.*

Property 6.3 *If $A \in \mathcal{C}$ and $A \subseteq Q$, then A is still a tight set.*

Property 6.4 *If $A \in \mathcal{C}$ and $Q \subseteq A$, then $Q'' \cup A$ is a tight set.*

Property 6.5 *There exists a tight set W such that $Q'' \subset W \subset (Q'' \cup R)$.*

The first three properties are easy to see. In the next section, we not only prove Property 6.5 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 6.5 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 Proof of Correctness

Suppose $Q \subset R$ are both tight sets in dynamic transshipment problem $(\mathcal{N}', v', T)$, and $|R \setminus Q| > 1$. Suppose source $s_0 \in R \setminus Q$ is chosen by an iteration of the algorithm. (The treatment of sinks is symmetric.) The iteration ultimately yields $Q' = Q \cup \{s_i\}$ and $Q'' = Q' \cup \{s_{i+1}\}$ and transit-time-parameterized problem $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$, which is feasible. Problem $(\overline{\mathcal{N}}^{\delta^*-1}, \overline{v}^{\delta^*-1}, T)$ is infeasible. Let $o(.)$ and $v(.)$ refer to feasible $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$, while $\tilde{o}(.)$ and $\tilde{v}(.)$ refer to infeasible $(\overline{\mathcal{N}}^{\delta^*-1}, \overline{v}^{\delta^*-1}, T)$.

Lemma 7.1 *If $\tilde{o}(A) < \tilde{v}(A)$ then $o(A) = v(A)$ and $s_{i+1} \in A$ but $s_0 \notin A$.*

Proof: Suppose $\tilde{o}(A) < \tilde{v}(A)$. Feasibility implies $o(A) \geq v(A)$. Decreasing the delay δ of source s_{i+1} cannot decrease the maximum dynamic flow out of any set: $\tilde{o}(A) \geq o(A)$. Furthermore, decreasing δ has no effect on $o(A)$ for any $A \not\ni s_{i+1}$; but a unit decrease of δ could increase $o(A)$

by at most one if $s_{i+1} \in A$. Applied to Q' and Q'', this means $v_{s_{i+1}} \leq \tilde{v}_{s_{i+1}} \leq v_{s_{i+1}} + 1$. Combined with the observation that no terminal supply other than $v_{s_{i+1}}$ can increase, the above inequalities yield

$$\tilde{v}(A) > \tilde{o}(A) \geq o(A) \geq v(A) \geq \tilde{v}(A) - 1.$$

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 $\tilde{v}(A) > v(A)$; because parameter δ changes v_{s_0} exactly opposite to $v_{s_{i+1}}$, this means that $s_{i+1} \in A$ but $s_0 \notin A$. □

Lemma 7.2 *If* $(A \cap R) \subseteq Q''$ *then* $\tilde{o}(A) \geq \tilde{v}(A)$.

Proof: We prove the lemma in two parts:

(1) *If* $A \subseteq Q''$ *then* $\tilde{o}(A) \geq \tilde{v}(A)$. Suppose $A \subseteq Q''$. By Lemma 7.1, if $s_{i+1} \notin A$ then $\tilde{o}(A) \geq \tilde{v}(A)$; thus we need only worry about the case when $s_{i+1} \in A$, which means $A \cup Q' = Q''$. Using submodularity with A and Q', we get $\tilde{o}(A) \geq \tilde{o}(Q'') + \tilde{o}(A \cap Q') - \tilde{o}(Q')$. Both Q' and Q'' are tight. Since $s_{i+1} \notin A \cap Q'$, Lemma 7.1 implies $\tilde{o}(A) \geq \tilde{v}(Q'') + \tilde{v}(A \cap Q') - \tilde{v}(Q') = \tilde{v}(A)$.

(2) *If* $(A \cap R) \subseteq Q''$ *then* $\tilde{o}(A) \geq \tilde{v}(A)$. 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 $\tilde{o}(A) \geq \tilde{o}(A \cup R'') + \tilde{o}(A \cap R'') - \tilde{o}(R'')$. Since $s_0 \in (A \cup R'')$, Lemma 7.1 implies $\tilde{o}(A \cup R'') \geq \tilde{v}(A \cup R'')$. Because we assume $(A \cap R'') \subseteq Q''$, the first part of the proof implies $\tilde{o}(A \cap R'') \geq \tilde{v}(A \cap R'')$. Property 6.4 implies $\tilde{o}(R'') = \tilde{v}(R'')$. Thus, we obtain $\tilde{o}(A) \geq \tilde{v}(A \cup R'') + \tilde{v}(A \cap R'') - \tilde{v}(R'') = \tilde{v}(A)$. □

Lemma 7.3 is a basic property of tight sets that follows from the submodularity of the $o(.)$ function:

Lemma 7.3 *In a feasible dynamic transshipment problem, the union and intersection of tight sets are tight.*

Let FEAS denote the time required to check feasibility of a dynamic transshipment problem, and to find a violated set if the problem is infeasible. (See Theorem 5.2.) The following theorem restates and proves Property 6.5:

Theorem 7.4 $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$ *contains a tight set of terminals* W *such that* $Q'' \subset W \subset (Q'' \cup R)$, *and* W *can be computed in* $O(\text{FEAS})$ *time.*

Proof: Let W' be any violated set in infeasible problem $(\overline{\mathcal{N}}^{\delta^*-1}, \overline{v}^{\delta^*-1}, T)$, so that $\tilde{o}(W') < \tilde{v}(W')$. By Lemma 7.1, W' is a tight set in $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$. Lemma 7.1 also implies $s_0 \notin W'$. Because $s_0 \in R \setminus Q$, this means $R \nsubseteq (Q'' \cup W')$. Lemma 7.2 implies $(W' \cap R) \nsubseteq Q''$.

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 6.2 and 6.4 and Lemma 7.3 imply that W is tight.

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

Theorem 7.5 *The dynamic transshipment problem can be solved in* $O(k \log(nUT)\text{FEAS})$ *time, and also in strongly polynomial time.*

Proof: The algorithm is dominated by the $O(k)$ iterations described in Section 6. Each iteration is dominated by the two binary searches. The first α-binary search tests feasibility each time it seeks integer $\alpha^* \in [0, nU]$. The second δ-binary search tests feasibility each time it seeks integer $\delta^* \in [0, T+1]$. To obtain the strongly polynomial time bound, replace each binary search by Megiddo's parametric search [28], as explained at the end of Section 5. $\Box$

To analyze the running time of the quickest transshipment algorithm, we bound the optimal time $T \leq n\mathcal{T} + V$ as described in Section 5, and we observe that the time to solve the dynamic transshipment problem $(\mathcal{N}, v, T)$ dominates the time to find the optimal time of the quickest transshipment problem $(\mathcal{N}, v)$.

Corollary 7.6 *The quickest transshipment problem can be solved in* $O(k \log(nUV\mathcal{T})\mathrm{FEAS})$ *time, and also in strongly polynomial time.*

8 Application and Extensions

In this section we describe an application of our quickest transshipment algorithm to network scheduling. We also discuss extensions of the dynamic transshipment problem that can be solved in polynomial time.

Network Scheduling. We considered network scheduling in Section 1. A network scheduling problem consists of a set of processors connected by links with capacities and transit times; each processor begins with a set of jobs of various sizes. The goal is to run all the jobs in the minimum overall time, where each job may be run by its home processor or may be sent into the network for remote execution. Here we consider the network scheduling problem when unit size jobs are distributed throughout a network of computers and we want to execute the last job as soon as possible. Based on a reduction to the integral quickest transshipment problem, we can compute an exact optimal schedule for this problem in polynomial time. This is the first efficient algorithm for general unit-job network scheduling.

Theorem 8.1 *The unit-job network scheduling problem can be solved via one quickest transshipment computation.*

Proof: Consider a unit-job network scheduling problem. Let V be the set of processors, connected by a set of links E, which are characterized by capacity and transit time functions u and τ. Let v_x be the number of unit-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. Each processor x in V is connected to t by a unit-capacity unit-transit-time edge. Let V' and E' denote the augmented graph, and likewise denote the extended capacity and transit time functions by u' and τ'. We define a supply function $v' : V' \to \mathbb{N}$ 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 function 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 job, and sending one unit of flow from some node x in V to supersink t corresponds to executing one job on processor x. $\Box$

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.

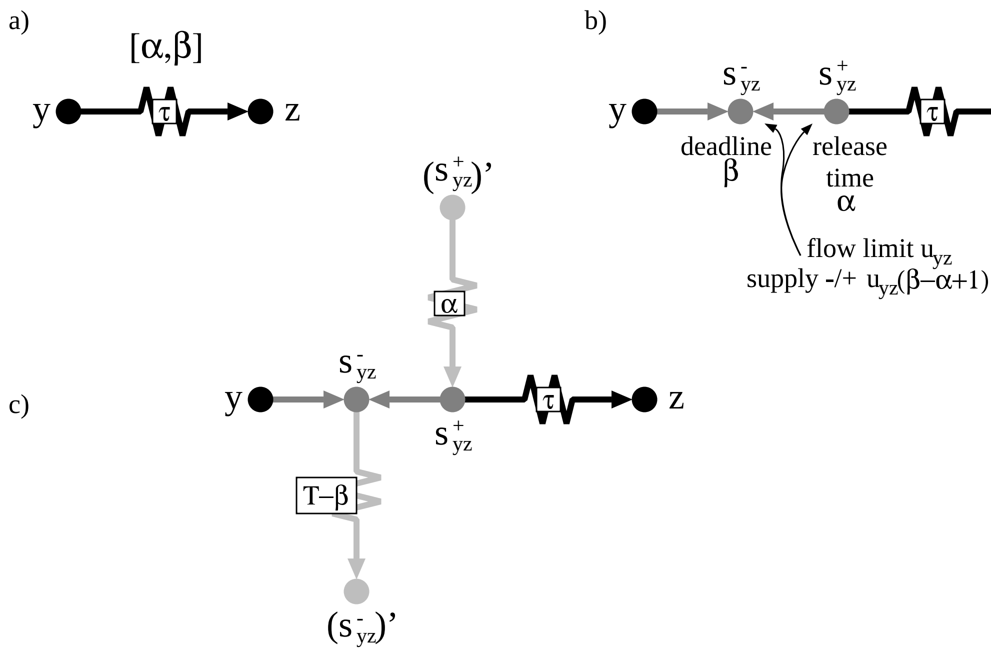

Figure 10: Reduction of a Mortal Edge to a Traditional Network

Networks with Release Times and Deadlines. We can extend our dynamic transshipment algorithm to handle more complicated networks consisting of sources with release times, sinks with deadlines, terminals with flow restrictions, and edges with start times and end times. These techniques also extend our quickest transshipment algorithm.

In a traditional dynamic transshipment problem, flow may leave each source vertex starting at time zero. We can generalize the problem and specify a release time function for sources; each source x cannot send flow before its specified release time r_x. This problem reduces to the original version simply by adding to the network a new set of sources S'; each new source x' in S' corresponds to an old source x in S; they are connected by an uncapacitated edge $x'x$ with transit time r_x. In this augmented network, allowing flow to leave x' at time zero is equivalent to holding flow at x until its release time r_x. In a similar manner, we can also allow a deadline function for sinks, so that each sink x cannot receive flow after its specified deadline d_x. We can also restrict the ability of sources to send flow or sinks to receive flow by augmenting the network as above but with capacitated edges.

We can further generalize a dynamic transshipment problem by specifying a start time α_{yz} and an end time β_{yz} for each edge yz in the network. 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.[1] We call interval $[\alpha_{yz}, \beta_{yz}]$ the *lifespan* of edge yz, and refer to yz as a *mortal edge*.

We can solve dynamic transshipment problems on networks built of mortal edges by adding extra terminals to the network. Figures 10(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, by introducing a new source and a new sink in the middle

[1]Holdover flow is permitted in the ordinary dynamic transshipment problem, but it is not required at any non-terminal vertex. If edges have start and end times, however, then holdover flow may be required throughout the network, and restricting the problem to flows without holdovers makes it NP-hard.

of every edge. Figure 10(c) further reduces the network to use terminals without release times, deadlines, or flow limits. Each mortal edge yz requires a new source s^+_{yz} with release time α_{yz} and a new sink s^-_{yz} with deadline β_{yz}; terminal s^+_{yz} (s^-_{yz}) can only send (receive) u_{yz} units of flow per time step. Then mortal edge yz reduces to three traditional edges: ys^-_{yz}, $s^+_{yz}s^-_{yz}$, and $s^+_{yz}z$. The capacity of each edge is u_{yz}. Transit time τ_{yz} is assigned to edge $s^+_{yz}z$, while ys^-_{yz} and $s^+_{yz}s^-_{yz}$ have zero transit time. Source s^+_{yz} has supply $u_{yz}(\beta_{yz}-\alpha_{yz}+1)$; sink s^-_{yz} has supply $-u_{yz}(\beta_{yz}-\alpha_{yz}+1)$; and the supplies of y and z remain unchanged.

Let $\mathcal{D}$ be a dynamic transshipment problem with release times, deadlines, and mortal edges. The above reduction yields a dynamic transshipment problem $\mathcal{D}'$ with release times, deadlines, and terminal flow restrictions, but without mortal edges. Note that the demand of the sink s^-_{yz} is the same as the supply of s^+_{yz} and so the total flow (over all times) on the edges $s^+_{yz}z$ and $ys^-_{y}z$ is the same. The amount of flow in each step is limited to be at most u_{yz} by the capacity, and release time and the deadline limite the flows into z and out of y to be at the right times. It is not immediate that the flow conservation is satisfied, i.e., that flow into node y arrives early enough to cross the edge yz in time before it leave from node z. The following theorem and proof imply that we can solve problem $\mathcal{D}$ by first solving problem $\mathcal{D}'$:

Theorem 8.2 *There is a feasible solution to $\mathcal{D}$ if and only if there is a feasible solution to $\mathcal{D}'$.*

Proof: Dynamic flow f must satisfy the supplies of all terminals, obey antisymmetry, conservation, and capacity constraints, and also respect source release times, sink deadlines, and edge lifespans. The only non-trivial constraints in this list are the conservation and capacity constraints.

Because we assume that dynamic flow f' respects conservation constraints, the conservation of dynamic flow f follows from two facts: the new terminals s^+_{yz} and s^-_{yz} have opposite (that is, balanced) supplies, and f may use holdover arcs at any node.

Holdover arcs allow flow to enter a node at one time and then leave that node some time later, resulting in a relaxed kind of flow conservation. However, the relaxation is restricted in one direction — a holdover arc must have non-negative flow. This restriction is a capacity constraint in the time-expanded graph.

The validity of holdover capacity constraints relies on the following property of dynamic flow f': for any time θ, the total dynamic flow value into s^-_{yz} before time θ is at least the total flow value out of s^+_{yz} before time θ. In addition to this property, we also use the fact the transit times of yz and $s^+_{yz}z$ are equal. The validity of non-holdover capacity constraints is trivial. $\square$

9 Summary and Open Problems

We introduce chain-decomposable flows in this paper. Chain-decomposable flows generalize two previously known techniques used in efficient dynamic network flow algorithms: stationary dynamic flows and temporally repeated flows. Stationary and temporally repeated flows have proven useful for solving infinite-horizon problems and the (single-source single-sink) maximum dynamic flow problem, but little else. Time-expanded networks do not lend themselves to efficient algorithms, but they are the predominant dynamic network flow model because of their general applicability. Chain-decomposable flows are flexible enough to model solutions for a much wider range of problems than stationary and temporally repeated flows; we also show how to compute them in polynomial time and thus improve on the pseudopolynomial results previously derived from time-expanded networks. Chain-decomposable flows lead to the first polynomial algorithms for the quickest transshipment problem, the dynamic transshipment problem, and the lexicographically maximum dynamic flow problem.

It is an interesting open problem to develop a purely combinatorial polynomial dynamic transshipment algorithm. For the moment, we test dynamic transshipment feasibility with a geometric algorithm that can optimize an arbitrary linear objective function over a convex set. A combinatorial algorithm would probably be more efficient (theoretically) and would almost certainly be more practical. 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. Second, Queyranne discovered a simple combinatorial algorithm to minimize *symmetric* submodular functions [33]. 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.

Acknowledgment

We are grateful to Bettina Klinz and Monika Rauch Henzinger for numerous discussions concerning dynamic flows. We are grateful to two anonymous referees for their careful reading of our paper, and for numerous suggestions for improving the writeup.

References

[1] Aronson, J.E., A Survey of Dynamic Network Flows, *Annals of Operations Research* 20(1989)1–66.

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

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

[4] Burkard, R.E., Dlaska, K. and Klinz, B., The Quickest Flow Problem, *ZOR Methods and Models of Operations Research* 37:1(1993)31–58.

[5] Chalmet, L.G., Francis, R.L. and Saunders P.B., Network Models for Building Evacuation, *Management Science* 28(1982)86–105.

[6] Choi,W., Francis,R.L., Hamacher,H.W., and Tufecki,S., 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.

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

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

[9] Deng, X., Liu, H., and Xiao, B., Deterministic Load Balancing in Computer Networks, *Proc. 2nd IEEE Symp. on Parallel and Distributed Processing*, 1992, 50–57.

[10] Fizzano, P. and Stein, C., Scheduling on a Ring with Unit Capacity Links, Dartmouth College Technical Report PC5-TR94-216.

[11] Fleischer, L., Faster algorithms for the quickest transshipment problem with zero transit times, to appear in *SIAM Journal on Optimization.*

[12] Fleischer, L., and Orlin J., Optimal rounding of fractional dynamic flows when transit times are zero, to appear in *SIAM Journal on Discrete Mathematics.*

[13] Fleischer,L. and Tardos,É., Efficient Continuous-Time Dynamic Network Flow Algorithms, *Operations Research Letters* 23:71-80, 1998.

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

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

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

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

[18] Hajek, B. and Ogier, R.G., Optimal Dynamic Routing in Communication Networks with Continuous Traffic, *Networks* 14(1984)457–487.

[19] Hamacher, H.W. and Tufekci, S., On the Use of Lexicographic Min Cost Flows in Evacuation Modeling, *Naval Research Logistics* 34(1987)487–503.

[20] Hoppe, B., Efficient Dynamic Network Flow Algorithms. Ph.D. Thesis, Cornell University. 1995. Department of Computer Science Technical Report TR95-1524.

[21] Hoppe, B. and Tardos, É., Polynomial Time Algorithms for Some Evacuation Problems, *Proc. 5th Annual ACM-SIAM Symp. Discrete Algorithms*, 1994, 433–441.

[22] Hoppe, B. and Tardos, É., The Quickest Transshipment Problem *Proc. 6th Annual ACM-SIAM Symp. Discrete Algorithms*, 1995, 512–521.

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

[24] Jarvis, J.R. and Ratliff, D.H., Some Equivalent Objectives for Dynamic Network Flow Problems, *Management Science* 28(1982)106–109.

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

[26] Klinz, B. Personal communication.

[27] Megiddo, N., Optimal Flows in Networks with Multiple Sources and Sinks, *Mathematical Programming* 7(1974)97–107.

[28] Megiddo, N., Combinatorial Optimization with Rational Objective Functions, *Mathematics of Operations Research*, 4(1979)414–424.

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

[30] Minieka, E., Maximal, Lexicographic, and Dynamic Network Flows, *Operations Research* 21(1973)517–527.

[31] Phillips, C., Stein, C., and Wein, J., Task Scheduling in Networks, *SIAM Journal on Discrete Mathematics*, 10(4):573–597, 1997.

[32] Powell, W.B., Jaillet, P. and Odoni, A., Stochastic and Dynamic Networks and Routing, in *Handbooks in Operations Research and Management Science, Networks*, (M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser, eds.), Elsevier Science Publishers B.V. (to appear).

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

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

[35] Vaidya, P.M., A New Algorithm for Minimizing Convex Functions over Convex Sets. *Proc. 30th Annual IEEE Symp. on Foundations of Computer Science*, 1989, 338–343.