Chapter 1
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 integral quickest transshipment problem. In an earlier draft of this paper [11], we gave a more complicated polynomial-time algorithm for the single-sink special case. Previously, the integral quickest transshipment problem could only be solved efficiently in the special case of a single source and single sink.

1 Introduction

Evacuating some structures seems harder than evacuating others. Young pupils in a one-room schoolhouse can simply scramble for the nearest exit when the final bell rings each day. Office workers in a modern skyscraper face a greater logistical challenge. If each occupant rushes greedily towards his nearest exit, the likely result is a set of congested bottlenecks, leaving some people inside for longer than truly necessary. Vacations can be even worse than the office, however. Passengers on a cruise ship face an even more complicated evacuation problem when their luxury liner founders on an iceberg. Not only must they leave their cabins and negotiate a network of narrow corridors, but they must also account for the finite capacities of the lifeboats that provide their only hope in the frigid waters. The problem is not just getting everyone off the cruise ship; it is getting everyone into a lifeboat in the minimum overall time.

In this paper we formalize these problems and describe the first polynomial-time algorithm that solves all of them. We use dynamic networks to model these problems. 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 $yz \in E$. In a feasible dynamic flow, at most u_{yz} units of flow can be pipelined along edge yz with each time step; flow leaving y at time θ reaches z at time $\theta + \tau_{yz}$.

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 integral quickest transshipment problem.

Dynamic networks typically 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, and furthermore, 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

†Department of Computer Science, Cornell University, Ithaca, NY 14853. Research supported by a National Science Foundation Graduate Research Fellowship.

‡School of Operations Research & Industrial Engineering, Cornell University, Ithaca, NY 14853. Research supported in part by a Packard Fellowship and an NSF PYI award.

been a fair amount of work in this area (see the surveys [1, 18]), 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.

Special cases of the quickest transshipment problem have been studied elsewhere. The evacuation problem is the quickest transshipment problem with a single sink;[1] it is commonly used to model building evacuation. Chalmet *et al* [3], Hamacher and Tufekci [9], and Jarvis and Ratliff [12] all studied pseudopolynomial algorithms for the evacuation problem. Hajek and Ogier [8] studied dynamic flows in networks with identically zero transit times; for this special case, they reduced the evacuation problem to n maximum flow computations, where n is the number of vertices in the network. In a previous paper [10], we described a polynomial-time algorithm for the (fractional) evacuation problem with a fixed number of sources. This paper describes the first polynomial-time algorithm for the integral evacuation problem (with a variable number of sources).

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 7), 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* [4] and Fizzano and Stein [5]. Deng *et al* [4] described a polynomial-time algorithm for unit-job network scheduling when every link is uncapacitated and has unit transit time. Recently, Fizzano and Stein [5] have 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* [17] 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.

[1] An earlier draft of this paper [11] defined the evacuation problem to require a flow that sends *at least* enough flow out of each source. The quickest transshipment problem requires a flow that sends *exactly* the right amount of flow out of each source. In a single-sink setting, the optimal times for these two problems are equal, and the problems are essentially equivalent.

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 [6] 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* [2] gave more efficient and strongly polynomial algorithms for this problem.

Problems in dynamic networks are equivalent to traditional network problems on an exponentially large time-expanded graph. Most known methods for solving practical dynamic network flow problems work directly on the time-expanded graph. These algorithms are pseudopolynomial: their running times are polynomial in the size of the graph G and the maximum allowed time T (rather than $\log T$).

Dynamic network flow problems are used to model real-time phenomena. Restricting the value of time bound T increases the granularity of the model and hence limits the approximation of reality. A pseudopolynomial dependence of running time on T thus forces a rough approximation of reality with large time steps. Designing polynomial algorithms for these problems allows for much greater precision in modeling.

After some definitions and preliminary discussion, Section 3 defines the lexicographic maximum dynamic flow problem (extending our original definition in [10]) and describes a polynomial-time algorithm for it. Section 4 describes how to compute the optimal time bound for a quickest transshipment problem. The next two sections contain the main result of this paper: Section 5 shows how to solve a quickest transshipment problem whose time bound is known, by reducing it to a lexicographic maximum dynamic flow problem; and Section 6 proves the correctness and running time of our algorithm. Finally, Section 7 applies our quickest transshipment algorithm to unit-job network scheduling and also discusses extensions of our main results.

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 integral *transit time* τ_{yz} associated with each edge $yz \in E$, and a set of *terminals* $S \subseteq V$. The maximum capacity is denoted by U and the maximum

transit time by $\mathcal{T}_m$. We also refer to transit times as *length* and *cost*. A terminal is either a source or a sink; the set of sources is denoted by S^+ and the set of sinks by S^-. We distinguish edges with positive capacity by including them also in set E^+. For simplicity, we make the following assumptions. E is symmetric: if $yz \in E$ then $zy \in E$; and τ is antisymmetric: if $yz \in E$ then $\tau_{yz} = -\tau_{zy}$. There are no parallel edges or zero length cycles in E. E^+ contains no opposite edges, and edges in E^+ have non-negative transit time. Sources have no entering edges in E^+, and sinks has no leaving edges in E^+. We use k to denote $|S|$; likewise, $n = |V|$ and $m = |E|$.

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 $y \in V$. Static flow or circulation f is *feasible* if it also satisfies capacity constraints $f_{yz} \leq u_{yz}$ for every edge yz. The *residual network* of the static flow or circulation f is defined as $\mathcal{N}_f = ((V,E), u^f, \tau, S)$, where the *residual capacity* function is $u^f_{yz} = u_{yz} - f_{yz}$. The *value* of a static (s,t)-flow f is $|f| = \sum_y f_{sy}$. Multi-source multi-sink static flows are defined analogously.

A *finite horizon dynamic flow* f with time bound T is equivalent to 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 $0 \leq \theta < T$. Dynamic flow f with time horizon T is a static flow in $G(T)$; we assume that f is identically zero before time 0 and after time T. An *infinite horizon dynamic flow* is equivalent to a static flow in the infinite time-expanded graph $G(*)$, defined analogously to $G(T)$. A dynamic flow is *feasible* if it satisfies the capacity constraints. The *dynamic value* of f is the net flow out of the source for all time steps, denoted by $|f|$. Multi-source multi-sink dynamic flows are defined analogously; in this case, $|f|_x$ denotes the net flow out of a particular terminal $x(\theta)$ for all time θ; note that for a sink x this is a non-positive amount.

The dynamic network flow problems in this paper are equivalent to easy (static) flow problems on the time-expanded graph; however, the size of the graph $G(T)$ is not polynomial when T is large.

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* is a multiple source and multiple sink version of the maximum dynamic flow problem. We are given a time bound T, and supplies v_x, where $v_x \geq 0$ for every source $x \in S^+$, and $v_x \leq 0$ for every sink $x \in S^-$. 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*, 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 is feasible. The *evacuation problem* is the special case of the quickest transshipment problem with a single sink.

Generalized Temporally Repeated Flows. As the value of time bound T grows, so does the size of time-expanded graph $G(T)$. Thus, dynamic network flow problems with large time bounds cannot be solved efficiently by ordinary static network flow computations on a time-expanded graph. Ford and Fulkerson [6] introduced temporally repeated flows to represent some simple dynamic flows efficiently. We define temporally repeated flows in this section and generalize the concept to allow efficient representation of a considerably larger class of dynamic flows; this generalization was originally introduced in [10].

A *chain flow* $\gamma = \langle v, P \rangle$ is a static flow of value $v \geq 0$ along path P in a network $\mathcal{N}$. If P is an (s,t)-path of length $\tau(\gamma)$, then given a time bound T no less than $\tau(\gamma)$, any feasible chain flow γ induces a feasible dynamic flow by sending v units of dynamic flow along path P every time step from time zero till time $T - \tau(\gamma)$. The last v units of flow finally reach the sink at time T.

Let $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ be a multiset of (s,t)-chain flows. We say that Γ is a *chain decomposition* of static (s,t)-flow f if $\sum_{i=1}^k \gamma_i = f$, and 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 Γ is a standard chain decomposition of f, every chain flow in Γ is no longer than time bound T, and f is feasible, then Γ induces a feasible dynamic flow, obtained by summing the dynamic flows induced by each chain flow in Γ.

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

Consider a standard chain decomposition Γ of static flow f. Ford and Fulkerson [6] observed that the dynamic value of $[\Gamma]^T$ depends only on f, and is independent of the choice of the standard chain decomposition Γ. The dynamic value can be expressed as

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

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 $-(T+1)$. Ford and Fulkerson 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.

We use *non-standard chain decompositions* to induce dynamic flows. The chain flows in a non-standard decomposition may use oppositely directed flows on arcs. For example, suppose network $\mathcal{N}$ consists of a single (s,t)-path P of capacity v. To this network, add a return arc ts with infinite capacity and length $-(T+1)$. Let C be the cycle consisting of P followed by ts, and let C' be the reverse of C (that is, st followed by the reverse of P). Define chain flows $\gamma = \langle v, C\rangle$ and $\gamma' = \langle v, C'\rangle$, so that $\gamma + \gamma'$ is the zero flow. Notice that dynamic flow $[\{\gamma, \gamma'\}]$ is equivalent to dynamic flow $[\{\langle v, P\rangle\}]^T$.

The dynamic flow induced by a non-standard chain decomposition may not be feasible, even if the sum of all chain flows is feasible. Consider the previous example. Chain flow γ' uses opposite edges of path P to cancel the flow induced by γ. If $(T+1)$ is less than the total length of path P, however, then the flow induced by γ' reaches each opposite edge before γ has provided any flow to cancel. The resulting dynamic flow is infeasible, even though $\gamma + \gamma'$ is the zero flow, which is feasible.

3 Lexicographic Maximum Dynamic Flows

In the *lexicographic maximum dynamic flow problem* we are given an ordering of the sources and sinks as $S = \{s_1, \ldots, s_k\}$ and we seek a feasible dynamic flow with specified time horizon T that lexicographically maximizes the amounts leaving the terminals in the order $s_1, \ldots, s_k$ (that is, for sinks lexicographically minimizes the amounts entering the sinks in this order). In our earlier paper [10] we considered the special case when the network has a unique sink t and t is last in the given order. In this section we show how the algorithm in [10] can be extended to solve the more general problem. The quickest transshipment problem will be solved in Sections 4, 5 and 6 by reducing it to the problem considered in this section.

Minieka [16] and Megiddo [14] 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 every i. Such a flow is clearly lexicographically maximum. This observation applies to the formulation of the lexicographic maximum 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.

Beginning with the zero flow, our lexicographic maximum dynamic flow algorithm computes successive layers of minimum cost (static) flows in the residual graphs of previous layers. Each layer yields a standard chain decomposition that is added to set Γ. At the end of the algorithm, $[\Gamma]$ is a lexicographic maximum dynamic flow.

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

The algorithm consists of k iterations indexed in descending fashion. In iteration $i = k-1, \ldots, 0$ we consider terminal s_{i+1}. If s_{i+1} is a sink, we add an infinite capacity arc $s_{i+1}s$ to G^{i+1} with transit time $-(T+1)$ to create graph G^i, and compute a minimum cost circulation f^i in the residual graph of the flow g^{i+1} in G^i. We add f^i to g^{i+1} to obtain flow g^i. If s_{i+1} is a source, then we delete the edge ss_{i+1} from G^{i+1} to create graph G^i, and compute a minimum-cost maximum flow f^i from s to s_{i+1} in the residual graph of the flow g^{i+1} in G^i. We add this minimum-cost flow f^i to the flow g^{i+1} to obtain flow g^i. Each flow f^i yields a standard chain decomposition. The assumption that there are no zero length cycles in G guarantees that there are no cycles in the decomposition. The chain flows are accumulated into chain decomposition Γ. Note that chain flows using an arc ss_i for a sink s_i

leave s_i at time $T+1$.

The proof of correctness of the algorithm proceeds essentially along the same lines as in [10]. We sketch the proof here, emphasizing the differences only.

The artificial edges and the assumption that sources have no entering edges in E^+ guarantee that after iteration i, the node s_{i+1} is balanced in the static flow g^i. This implies the following lemma:

Lemma 3.1 For any iteration i, static flow g^i is a minimum cost circulation in graph G^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 s to vertex $x \in V$ in the residual graph of the flow g^i in G^i.

Lemma 3.2 For any vertex $x \in V$ and any iteration i, $p^i(x) \geq p^{i+1}(x)$.

Proof: Every iteration computes a minimum-cost flow from node s to some node s_{i+1} in the residual graph of a previously computed flow. If s_{i+1} is a source this is true by definition, if s_{i+1} is a sink, then the minimum-cost circulation f^i is a flow through the arc $s_{i+1}s$ added in this iteration. The only edges added to the network are entering s, and the residual graphs have no negative cycles. This implies the lemma. □

Theorem 3.3 $[\Gamma]$ is a feasible dynamic flow.

Proof: In [10] it was shown that Lemma 3.2 implies that the chain flows obtained from flow f^i reach every edge in G after the chain flows obtained from flow f^j for every $j > i$. This implies by induction on the number of iterations that the resulting dynamic flow does not violate the capacity constraints. To see that at no time do sources have net incoming flow, nor sinks have net outgoing flow, we use the assumption that sources have no entering edges and sinks have no leaving edges in E^+. □

Theorem 3.4 $[\Gamma]$ is a lexicographic maximum dynamic flow.

Proof: Our proof relies on the infinite time-expanded graph $G(*)$. Given any index $i : 1 \leq i \leq k$, we construct a cut C_i in the time-expanded graph, and show that $[\Gamma]$ saturates C_i. The 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 $C_i = \{x(\theta) : \theta \geq p^i(x)\} \cup \{s_j(T) : j \leq i, s_j \in S^-\}$. The artificial edges ss_j for $j \leq i$ imply that C_i contains the source set. To see that C_i is disjoint from the sink set, we use the fact that the residual graph has no negative cycles, therefore $p^i(s_j) \geq T+1$ for sink s_j such that $j > i$; and we use the fact that there are no edges in E^+ entering sources, therefore $p^i(s_j) = +\infty$ for sources s_j such that $j > i$.

The fact that $[\Gamma]$ saturates the cut C_i can be shown along the same lines as the analogous theorem in [10] using in addition the assumption that there are no edges in E^+ leaving the set S^-. □

Dynamic flow $[\Gamma]$ appears to be an infinite horizon dynamic flow, and so in Theorem 3.4 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. The proof is similar to one given in [10].

Theorem 3.5 $[\Gamma]$ has time horizon T.

Theorem 3.6 A lexicographic maximum dynamic flow with k sources and sinks can be computed via k minimum cost flow computations.

4 Testing 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. Let MCF denote the time of a single minimum cost flow computation; then $o(A)$ can be determined in $O(\text{MCF})$ time. 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 [13] 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 *tight* if $o(A) = v(A)$, and *violated* if $o(A) < v(A)$.

Theorem 4.1 (Klinz [13]) 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 [14]).

Theorem 4.2 A violated set in a dynamic transshipment problem $(\mathcal{N}, v, T)$ can be found in $O(2^k \,\mathrm{MCF})$ time, in $O(k^2 \,\mathrm{MCF} \log(nTU))$ time[2], and also in strongly polynomial time.

Proof: The $O(2^k \,\mathrm{MCF})$ 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 [7]. For the $O(k^2 \,\mathrm{MCF} \log(nTU))$ time version, we use Vaidya's algorithm [19]. We omit details here. □

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 4.2.

To analyze the binary search, we need an upper bound for the optimal time T. Theorem 4.1 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 $n\mathcal{T}_{\mathrm{m}}$. 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 $n\mathcal{T}_{\mathrm{m}}$ time steps, at least one unit of flow can reach the sink with each time step, until there is no more flow. Thus the optimal time for the quickest transshipment problem can be bounded: $T \leq n\mathcal{T}_{\mathrm{m}} + V$.

We can also find the optimal time T in strongly polynomial time using Megiddo's parametric search [15] instead of the binary search.

[2] To simplify this and subsequent running times, we assume $k \,\mathrm{MCF} \geq$ the time required for a $k \times k$ matrix inversion.

Theorem 4.3
The optimal time T for the quickest transshipment problem can be found in $O(2^k \,\mathrm{MCF} \log(nV\mathcal{T}_{\mathrm{m}}))$ time, in $O(k^2 \,\mathrm{MCF} \log^2(nUV\mathcal{T}_{\mathrm{m}}))$ time, 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.

5 Integral Dynamic Transshipment Algorithm

Overview of Algorithm. Our algorithm reduces any dynamic transshipment problem $(\mathcal{N}, v, T)$ to an equivalent lexicographic maximum dynamic flow problem on a slightly more complicated network $\mathcal{N}'$. We described an algorithm to solve the lexicographic maximum dynamic flow problem in Section 3.

Let S be the set of terminals in the original network $\mathcal{N}$. Our algorithm attaches a new set of terminals S' to S. Initially, S' contains one source s_0 for each source $s \in S$, connected by edge $s_0 s$ with infinite capacity and zero transit time; likewise, S' contains one sink t_0 for each sink $t \in S$, connected by edge tt_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 $(\mathcal{N}, v, T)$.

The body 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 x_0 so that for any $x \in S$, the total supply of terminals in S' associated with x is v_x. This maintains the invariant that any solution to $(\mathcal{N}', v', T)$ yields a solution to $(\mathcal{N}, v, T)$.

The algorithm maintains a chain $\mathcal{C}$ whose elements are tight subsets of S' ordered by inclusion. The goal of the algorithm is to extend $\mathcal{C}$ to a complete chain. The following theorem implies that a complete chain $\mathcal{C}$ reduces the associated dynamic transshipment problem to a single lexicographic maximum dynamic flow problem on the same network:

Theorem 5.1 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 can be solved by a single lexicographic maximum dynamic flow in $\mathcal{N}'$.

Proof: If $|\mathcal{C}| = |S'| + 1$, then there is an ordering $S' = \{s_1, \ldots, s_{k'}\}$ 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 order of chain $\mathcal{C}$. Conversely, this also means that any solution to the lexicographic maximum 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 so $|\mathcal{C}| - |S'| = 2 - k$. By the above theorem, we are done when $|\mathcal{C}| - |S'| = 1$, and so we need to increase the value of this expression. Below, we describe how one iteration of the algorithm increases $|\mathcal{C}| - |S'|$ by one and maintains the feasibility of $(\mathcal{N}', v', T)$. Thus, the given dynamic transshipment problem is reduced to an equivalent lexicographic maximum 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 $|R \setminus Q| > 1$. (By adjacent we mean $\nexists A \in \mathcal{C} : Q \subset A \subset R$.) If no such Q and R exist, then $|\mathcal{C}| - |S'| = 1$ and we are done. The algorithm maintains the invariant that every terminal in $R \setminus Q$ is one of the first terminals x_0 in S'.

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 we use the following steps to increase $|\mathcal{C}| - |S'|$ by one.

Source s_0 is adjacent to s, one of the original sources in S; suppose there are $i-1$ other sources in S' adjacent to s. Add a new source s_i to S', 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. Define new supply v^α: the supply of s_i is $v^\alpha_{s_i} = o(Q') - o(Q)$; the supply of s_0 is reduced by $v^\alpha_{s_i}$; and other terminal supplies remain unchanged.

Consider the above parameterized dynamic transshipment problem $(\mathcal{N}^\alpha, v^\alpha, T)$. Notice that if $\alpha = 0$, then the maximum flow out of s_i is zero: the problem is equivalent to $(\mathcal{N}', v', T)$ and is therefore feasible. At the opposite extreme, if $\alpha = \infty$ then $(\mathcal{N}^\alpha, v^\alpha, T)$ must be infeasible — if it were feasible then $Q \cup \{s_0\}$ must be a tight set, but we have already determined that $Q \cup \{s_0\}$ is not tight. 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 $s \in S$.

Consider the dynamic transshipment problem $(\mathcal{N}^{\alpha^*}, v^{\alpha^*}, T)$. In this network, add another source s_{i+1} to S', connected to s by a parameterized δ-length unit-capacity arc. Let $Q'' = Q' \cup \{s_{i+1}\}$. Define new supply $\overline{v}^\delta$: the supply of s_{i+1} is $\overline{v}^\delta_{s_{i+1}} = o(Q'') - o(Q')$; the supply of s_0 is reduced by $\overline{v}^\delta_{s_{i+1}}$; and other terminal supplies remain unchanged.

Now we have a new parameterized dynamic transshipment problem $(\overline{\mathcal{N}}^\delta, \overline{v}^\delta, T)$. Notice that if $\delta = T+1$, then the maximum flow out of s_{i+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.

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

Property 5.2 Q' and Q'' are both tight sets.

Property 5.3 If $A \in \mathcal{C}$ and $A \subseteq Q$, then A is still a tight set.

Property 5.4 If $A \in \mathcal{C}$ and $Q \subseteq A$, then $Q'' \cup A$ is a tight set.

Property 5.5 There exists a tight set W such that $Q'' \subset W \subset (Q'' \cup R)$.

The first three properties are easy to show. In the next section, we not only prove Property 5.5 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 5.5 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.

6 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 $o'(.)$ and $v'(.)$ refer to infeasible $(\overline{\mathcal{N}}^{\delta^*-1}, \overline{v}^{\delta^*-1}, T)$.

Lemma 6.1 If $o'(A) < v'(A)$ then $o(A) = v(A)$ and $s_{i+1} \in A$ but $s_0 \notin A$.

Proof: Suppose $o'(A) < v'(A)$. 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: $o'(A) \geq o(A)$. 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 $v_{s_{i+1}} \leq v'_{s_{i+1}} \leq v_{s_{i+1}} + 1$. Combined with the observation that no terminal supply other than $v_{s_{i+1}}$ can increase, the above inequalities yield

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

Notice that all but the first element of this chain must in fact be equal, so that A is a tight set: $o(A) = v(A)$. This chain also shows $v'(A) > v(A)$; because parameter 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 6.2 If $(A \cap R) \subseteq Q''$ then $o'(A) \geq v'(A)$.

Proof: We prove the lemma in two parts:

(1) *If $A \subseteq Q''$ then $o'(A) \geq v'(A)$.* Suppose $A \subseteq Q''$. By Lemma 6.1, if $s_{i+1} \notin A$ then $o'(A) \geq v'(A)$; 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 $o'(A) \geq o'(Q'') + o'(A \cap Q') - o'(Q')$. Both Q' and Q'' are tight. Since $s_{i+1} \notin A \cap Q'$, Lemma 6.1 implies $o'(A) \geq v'(Q'') + v'(A \cap Q') - v'(Q') = v'(A)$.

(2) *If $(A \cap R) \subseteq Q''$ then $o'(A) \geq v'(A)$.* Let $R'' = (Q'' \cup R)$. Suppose $(A \cap R) \subseteq Q''$; this implies $(A \cap R'') \subseteq Q''$. Using submodularity with A and R'', we get $o'(A) \geq o'(A \cup R'') + o'(A \cap R'') - o'(R'')$. Since $s_0 \in (A \cup R'')$, Lemma 6.1 implies $o'(A \cup R'') \geq v'(A \cup R'')$. Because we assume $(A \cap R'') \subseteq Q''$, the first part of the proof implies $o'(A \cap R'') \geq v'(A \cap R'')$. Property 5.4 implies $o'(R'') = v'(R'')$. Thus, we obtain $o'(A) \geq v'(A \cup R'') + v'(A \cap R'') - v'(R'') = v'(A)$. □

Lemma 6.3 is a basic property of tight sets that follows from the submodularity of the $o(.)$ function:

Lemma 6.3 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. (See Section 4.) The following theorem restates and proves Property 5.5:

Theorem 6.4 $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$ 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 $o'(W') < v'(W')$. By Lemma 6.1, W' is a tight set in $(\overline{\mathcal{N}}^{\delta^*}, \overline{v}^{\delta^*}, T)$. Lemma 6.1 also implies $s_0 \notin W'$. Because $s_0 \in R \setminus Q$, this means $R \not\subseteq (Q'' \cup W')$. Lemma 6.2 implies $(W' \cap R) \not\subseteq Q''$.

Consider $W = Q'' \cup (W' \cap R)$. The above statements imply that $Q'' \subset W \subset (Q'' \cup R)$. Furthermore, after rewriting W as $(Q'' \cup W') \cap (Q'' \cup R)$, observe that Properties 5.2 and 5.4 and Lemma 6.3 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 6.5 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 5. 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 [15]. □

To analyze the running time of the quickest transshipment algorithm, we bound the optimal time $T \leq n\mathcal{T}_{\mathrm{m}} + V$ as described in Section 4, 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 6.6 The quickest transshipment problem can be solved in $O(k \log(nUV\mathcal{T}_{\mathrm{m}})\mathrm{FEAS})$ time, and also in strongly polynomial time.

7 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 defined network scheduling in Section 1. Various network scheduling problems have been studied by Deng *et al* [4], Phillips *et al* [17], and Fizzano and Stein [5].

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 7.1 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 zero 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. □

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.

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.[3] We call interval

[3] 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.

$[\alpha_{yz}, \beta_{yz}]$ 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. Each mortal edge yz requires a new source x^+_{yz} with release time α_{yz} and a new sink x^-_{yz} with deadline β_{yz}; terminal x^+_{yz} (x^-_{yz}) can only send (receive) u_{yz} units of flow per time step. Then mortal edge yz reduces to three traditional edges: yx^-_{yz}, $x^+_{yz}x^-_{yz}$, and $x^+_{yz}z$. The capacity of each edge is u_{yz}. Transit time τ_{yz} is assigned to edge $x^+_{yz}z$, while yx^-_{yz} and $x^+_{yz}x^-_{yz}$ have zero transit time. Source x^+_{yz} has supply $u_{yz}(\beta_{yz} - \alpha_{yz} + 1)$; sink x^-_{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. Notice that this reduction is very similar to the standard technique used to transform a capacitated minimum cost flow problem into an uncapacitated minimum cost flow problem. The following theorem and proof sketch imply that we can solve problem $\mathcal{D}$ by first solving problem $\mathcal{D}'$:

Theorem 7.2 There is a feasible solution to $\mathcal{D}$ if and only if there is a feasible solution to $\mathcal{D}'$.

Proof sketch: Given a solution to problem $\mathcal{D}$, it is not hard to produce a solution to problem $\mathcal{D}'$. To see the other direction, let f' be a feasible dynamic flow that solves $\mathcal{D}'$. For any mortal edge yz in the network of $\mathcal{D}$, define $f_{yz}(\theta)$ to be $u_{yz} - f'_{x^+_{yz}x^-_{yz}}(\theta)$ for any time θ in the lifespan of yz, or zero otherwise. Then f is a feasible dynamic flow that solves $\mathcal{D}$.

Acknowledgement

We are grateful to Monika Rauch for numerous discussions concerning integral dynamic flows.

References

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

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

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

[4] 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.

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

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

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

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

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

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

[11] Hoppe, B. and Tardos, É., A Polynomial Time Algorithm for the Integral Evacuation Problem, unpublished manuscript, November 1993.

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

[13] Klinz, B. Personal communication.

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

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

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

[17] Phillips, C., Stein, C., and Wein, J., Task Scheduling in Networks, *Proc. Fourth Scandinavian Workshop on Algorithm Theory*, 1994, 290–301.

[18] 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).

[19] 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.