A Faster Algorithm for the Quickest Transshipment Problem

L. Fleischer[*]

December 20, 1996

Working paper. Please do not circulate.

Abstract

A transshipment problem with demands that exceed network capacity can be solved by sending flow in several waves. How can this be done using the minimum number of iterations? This is the question tackled in the quickest transshipment problem. Hoppe and Tardos [8] describe the only known polynomial time algorithm that finds an integral solution to this problem. Their algorithm repeatedly minimizes submodular functions using the ellipsoid method, and is therefore not at all practical. I present an algorithm that finds a fully integral quickest transshipment with a polynomial number of maximum flow computations. When there is only one sink, the quickest transshipment problem is significantly easier. For this case, I show how the algorithm can be sped up to return an integral solution using $O(k)$ maximum flow computations, where k is the number of sources.

Hajek and Ogier [7] describe an algorithm that finds a fractional solution to the single-sink quickest transshipment problem on a network with n nodes using $O(n)$ maximum flow computations. They actually solve the universally quickest transshipment—a dynamic flow that minimizes the amount of supply left in the network at every moment of time. In this paper, I show how to solve this problem in the same asymptotic time as one maximum flow computation.

---

[*] email: lisaf@orie.cornell.edu. School of Operations Research and Industrial Engineering, Cornell University. Supported in part by an AAUW Selected Professions Engineering Dissertation Fellowship and by AASERT through grant N00014-95-1-0985.

1 Introduction

The field of network flows blossomed in the 1940s and 50s with interest in transportation planning, and has developed rapidly since then. There is a significant body of literature devoted to this subject. However, it has largely ignored a crucial aspect of transportation: transportation occurs over time. In the 1960's, Ford and Fulkerson introduced dynamic network flows to include time in the network model. Since then, there has been a fair amount of study devoted to this topic, for example see the surveys of Aronson [3] and Powell, et al. [13].

Dynamic network flows are used to model not only vehicular transportation, but also electronic communication and job scheduling. A dynamic network consists of a graph with capacities and transit times on the edges. Flow moves through a dynamic network over time. Edge capacities restrict the rate of flow and edge transit times determine how long each unit of flow spends traversing the network.

In this paper, I discuss a special case of dynamic network flow problems: I assume all transit times are zero. Dynamic network flow models with zero transit times lie between general dynamic flow models and traditional networks. They have been considered before, for instance, in the book of Anderson and Nash [1]. They capture some time-related issues: dynamic networks without transit times can be used to model instances when network capacities restrict the quantity of flow that can be sent at any one time. In this paper, I describe algorithms to solve the *quickest transshipment problem*—minimize the time necessary to solve a transshipment problem in a network with more demand than capacity. The solution in this special case may help in finding a more efficient exact or approximate algorithm to solve harder dynamic network problems.

When there is only one source and sink, the quickest transshipment problem is a quickest flow problem, and is solved easily with one maximum flow computation: the supply at the source divided by the capacity of the minimum source-sink cut in the network determines the quickest time T^* in which it is possible to satisfy all supplies. The feasible flow completing by this time is obtained by sending the maximum flow amount from time 0 until time T^*. Unlike static flows, the multiterminal dynamic transshipment problem is not reducible to a single source, single sink dynamic flow problem: sending a maximum amount of flow from sources to sinks may result in sending too much flow from some sources or to some sinks, and not enough from or to other terminals.

A fractional solution for a dynamic transshipment problem given time bound T is easy to find, and I will show this can be done with a single maximum flow. Hajek and Ogier [7] describe an algorithm that finds a fractional solution to the quickest transshipment problem with the additional property that it minimizes the amount of excess left in the network at every moment of time. This is called a *universally quickest transshipment.* Hajek and Ogier claim a run time that uses $O(n)$ maximum flow computations. I will describe how this problem can be solved in time asymptotic to one maximum flow computation. There is a two source, two sink example for which there is no universally quickest transshipment.

A further challenge is to find a solution that has integral flow rates for discrete time units. Many of the applications of dynamic flows need integral solutions: when flows are of big objects, like airplanes, or train engines, the amounts are often small, so fractional approximations are not very useful. Recently, Hoppe and Tardos [8] describe the only known polynomial time algorithm to solve this problem. They actually solve the harder problem in which arcs have non-zero transit times. However, their algorithm repeatedly calls the ellipsoid method as a subroutine, so it is not practical, and is much slower theoretically.

I present a new framework for solving quickest transshipment problems. This framework allows me to find integral solutions quickly. In the special case when there is only one sink, I solve the integral quickest transshipment problem with $O(k)$ maximum flows, where k is the number of terminals. In the general, multisource, multisink case, I show that the integral quickest transshipment can be solved with $O(k^2 \log Un + k \log \Sigma)$ maximum flow computations, where U is the maximum capacity in the graph, Σ is the sum of all positive supplies, and k is the number of terminals. I also describe a strongly polynomial time algorithm.

While the solving a transshipment problem with fixed supplies is useful for clearing a network after a communication breakdown, every day usage more often involves continuous streams of traffic. All of the algorithms presented here, like the algorithm of Hajek and Ogier [7], allow for constant streams of flow into or out of any node in the network.

The model as presented in this paper can also allow storage at any node of the network. But the algorithms will find a dynamic flow that depletes all supplies in the minimal time, and does not ever increase the absolute value of the supply at any node. That is, the algorithm never uses any additional storage.

Traditional approaches to solving the dynamic transshipment problem measure time in discrete steps and make use of the time expanded graph of the original network [3, 13]. A *time-expanded network* is a directed graph that contains a copy of the network for every time step, and *holdover arcs* from a copy of a node at time θ to the copy of the same node at time $\theta + 1$. In the time-expanded network, the dynamic transshipment problem is a traditional static transshipment problem with the set of sources composed of sources in the first copy of the network, and the set of sinks consisting of the sinks in the final copy of the network. Unfortunately, this graph may be very large, and is thus not practical to work with for large discretizations. Hoppe and Tardos [8] describe the only polynomial time algorithm to solve the discrete problem. However, it repeatedly calls the ellipsoid algorithm as a subroutine, and hence is also not practical.

Dynamic network flows have also been considered in the continuous time setting [1, 2, 12, 14]. Here, capacities are upper bounds on the rate of flow through the arcs. Most of the work in this area has examined networks with time-varying edge capacities, storage capacities, or costs. The focus of this research is on proving the existance of optimal solutions for classes of time-varying functions, and proving the convergence of algorithms that eventually find solutions. These algorithms fall short of being efficient, either theoretically or practically, and implementations do not seem able to handle problems with more than a few nodes. For the case in which capacity functions are constant, Fleischer and Tardos [4] extend the polynomial, discrete-time dynamic transshipment algorithm of Hoppe and Tardos [8] to work in the continuous-time setting.

The integral quickest transshipment algorithm presented in this paper returns both continuous time and discrete time solutions.

2 Notation

A network $\mathcal{N} = (V, E, u, S)$ has vertex set V of cardinality n, and arc set E of cardinality m. Arc e has capacity u_e and $U = max_e u_e$. S is a subset of the nodes containing *terminals*: nodes with non-zero supplies. The sum of supplies over all terminals equals zero. Let S^+ be the set of terminals, also called *sources*, with positive supply, and S^- be the set of terminals, also called *sinks*, with negative supply. Σ is defined as the sum of all positive supply. The sum of all supply in a set of nodes $A \subset V$ is denoted

$\sigma(A)$, with $\Sigma = \sigma(S^+)$. Given A, let $o(A)$ denote the amount of flow that can be sent from sources inside A to sinks outside A in one unit of time. Note that, since all transit times are zero, the amount of flow that can be sent from A to outside A in time T is $o(A)T$.

Network $\mathcal{N}' = (V', E')$ is a new network with vertex set $V' := V \cup \{s, t\}$ and edge set $E' := E \cup \{(s, i) | i \in S^+\} \cup \{(j, t) | j \in S^-\}$. Here s and t are referred to as the super-source and the super-sink respectively.

3 Dynamic transshipment feasibility

All algorithms described in this paper rely on testing dynamic transshipment feasibility. In the problem with transit times, Hoppe and Tardos [8] use submodular function minimization to resolve dynamic transshipment feasibility in time T. Zero transit times make the problem much easier. Let S^+ be the set of all terminals with positive supply, S^- the set of all terminals with negative supply. Given network $\mathcal{N} = (V, E, u, S)$ and supplies σ, construct static network $(\mathcal{N}', u')$ where $u'_{ij} = u_{ij}T$, $(i, j) \in E$, $u'_{si} = \sigma_i$, $u'_{jt} = -\sigma_j$.

Lemma 3.1 *The dynamic transshipment problem is feasible in time T if and only if $\{s\}$ is a minimum (s, t)-cut in $(\mathcal{N}', u')$.*

Proof: Flow on any edge summed over the course of a feasible dynamic transshipment cannot exceed T times the capacity of the edge. If $\{s\}$ is a minimum (s, t)-cut in this network, then the maximum (s, t)-flow f has value equal to the sum of all supplies. This flow can be transformed into a feasible dynamic transshipment by sending flow at rate f_{ij}/T through each arc (i, j) from time 0 until time T. ∎

Theorem 3.2 *Given time bound T, a fractional dynamic transshipment can be found with one maximum flow computation.*

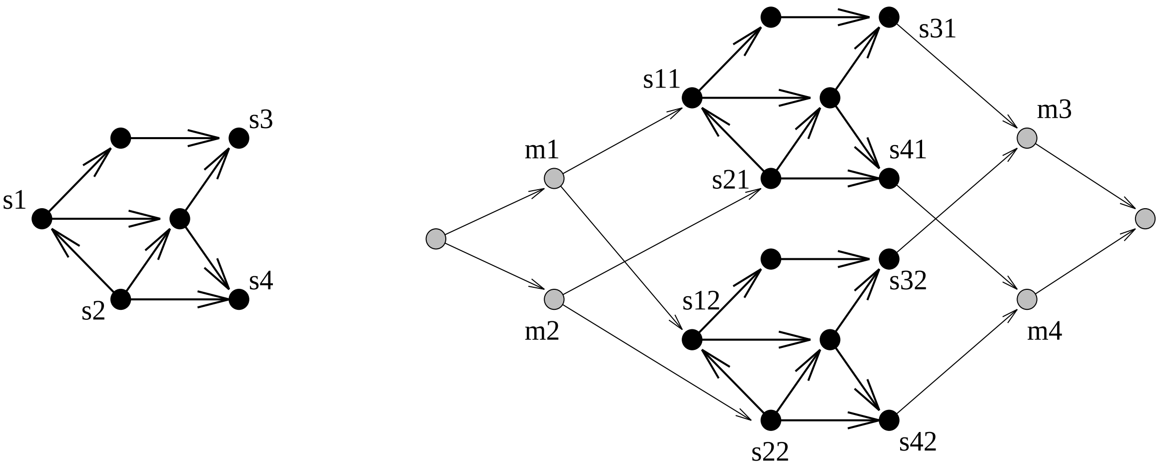
Binary search can be used to find the minimum time T^* needed for dynamic transshipment feasibility. This T^* is defined by some cut A in the network with the property that $o(A)T^* = \sigma(A)$. Any set A with this property is called *tight*. T^* may be fractional, since $\sigma(A)$ may not be an integral multiple of $o(A)$; but, since all supplies and capacities are integers, it's denominator, at most $o(A)$, will be bounded by the size d of the minimum cut in $\mathcal{N}'$ when $T = 1$. If the dynamic transshipment problem is feasible for some time T, then it is feasible for $T = \Sigma$. We can then find the optimal time T^* with $O(\log dT^*) = O(\log nU\Sigma)$ maximum flow computations.

Theorem 3.2 implies that the maximum flow f for optimal time T^* yields an optimal fractional transshipment: sending flow at rate f_{ij}/T^* through each arc (i, j) from time 0 until time T^* depletes all supplies by time T^*. A further challenge is to find a solution in which the flow rates (or flow amounts, in the discrete time case) are integers. Since the dynamic transshipment problem is equivalent to a static transshipment in the time expanded graph, we know an integral solution exists.

4 Quickest integral transshipment

I first describe an algorithm to solve the quickest integral, continuous-time transshipment problem with $O(k^2 \log Un + k \log \Sigma)$ maximum flow computations, or a strongly polynomial number of max-

Figure 1: A network, and the corresponding two-level network



imum flow computations. This produces a solution with constant, integral rates of flow over time periods of arbitrary intervals. I next explain how the continous-time solutions constructed by the algorithm can transformed into a discrete-time integral transshipment with at most k more maximum flow computations. When there is only one sink, a slightly modified algorithm solves the integral quickest transshipment problem with only $O(k)$ maximum flow computations.

4.1 Quickest continuous-time integral transshipment

To find an integral dynamic transshipment, the algorithm uses a network inspired by the time-expanded network, called a *two-level network* (Figure 1). The two-level network contains two copies of the original network. One copy represents the first unit of time and has original arc capacities. The other copy represents the remaining time, and has capacities multiplied by $T^* - 1$. In addition, the two-level network contains $k + 2$ new vertices: master-terminals m_i, $i = 1$ to k, a super-source, and a super-sink. Let s_i^1 be the copy of terminal s_i in the small capacity network copy, and s_i^2 be the corresponding terminal in the large capacity network copy. Each master-terminal, m_i is connected to s_i^1 and s_i^2 by infinite capacity arcs. A super-source is connected to each master-source by an arc of capacity equal to the supply of the source, and each master-sink is connected to the super-sink by an arc of capacity equal to the demand at the sink.

A maximum flow computation in the two-level network gives an integral flow, since all inputs are integral. The flow in the small capacity copy of the original network corresponds to the flow sent in the first unit of time. The flow in the large capacity component indicates that the remaining supplies can be satisfied in time $T^* - 1$. That is, this reduced dynamic transshipment problem with reduced supplies $f_{m_i s_i^2} = |\sigma_i| - f_{m_i s_i^1}$ is feasible in time $T^* - 1$.

If we construct a new two-level graph for this reduced problem, and repeat the process, we then find an integral flow for the second unit of time, and a further reduced problem feasible in time $T^* - 2$. This leads to the following observation.

Observation 4.1 *Given feasible time bound T, an integral dynamic transshipment can be found with $T - 1$ maximum flow computations.*

To develop a more efficient algorithm, it is necessary to reduce the number of maximum flow computations. Let f be the portion of the flow using the small capacity part of the 2-level network, and let f_i be the amount of flow passing through terminal s_i^1, eg. $f_i = f_{m_i s_i^1}$ if s_i is a source. We would like to continue to use f for as long as the remaining supplies $|\sigma| - \lambda f$ can be satisfied in the remaining time, $T^* - \lambda$. Finding the maximum λ so that the remaining problem is feasible is equivalent to solving a parametric flow problem in a static network. Let $\mathcal{N}^\lambda = (V', E', u^\lambda)$ where u^λ equals $\sigma_i - \lambda f_i$ on edges leaving the supersource, $-\sigma_j - \lambda f_j$ on edges entering the supersink, and $u_{ij}(T^* - \lambda)$ on all other edges (i, j). Here, λ is the amount of time we can use the first integral flow. Starting with $\lambda = 1$, we can increase λ until either the supply is depleted at some terminal (eg. $\sigma_i - \lambda f_i = 0$ for some i), or until the reduced problem becomes infeasible. The reduced problem remains feasible as long as $\{s\}$ is a minimum cut in $\mathcal{N}^\lambda$. When this is no longer true, there is some subset of nodes $A \subset S$ that are also contained in the minimum cut. The supply in A exceeds the capacity of the arcs leaving A. Hence, the remaining supply at A cannot empty from A in the remaining time. Because the capacities are linear functions of λ, increasing λ further will not correct this problem. Hence, once the reduced problem becomes infeasible, it remains infeasible; so we can binary search for the maximum feasible λ, with $1 \leq \lambda \leq min_i \frac{|\sigma_i|}{f_i}$.

The λ we find may be fractional, but its denominator d is bounded by the size of a cut in the original network, which is bounded by Um. Once we find λ, we can remove the supply sent during time λ, and focus our attention on the reduced problem. This problem now has a time bound that may be fractional, as well as supplies that may be fractional. By multiplying the new supplies $|\sigma| - \lambda f$ and the new time bound $T^* - \lambda$ by d, we obtain an integral problem that is equivalent to the fractional problem: Any flow g for the original problem can be transformed into a flow satisfying the multiplied problem by sending each amount of flow for a period that is d times longer than the time it is sent in the original problem. So the multiplied problem is feasible if the original problem is. Since all its input data are integral, it has an integral solution. We can then take that integral solution and use it to find an integral solution to the original problem by reducing the *time* any particular flow is sent by a factor of d.

Figure 2 summarizes the algorithm. The quickest integral transshipment takes as input a network with arc capacity vector U, node supply vector σ, and time bound T, and returns a set of flow-duration pairs that specify the flows that compose the optimal transshipment and the duration of time for which they continue.

The algorithm repeats this procedure until it exhausts all supplies. How many repetitions are necessary? At most k. Each time we fix a λ, one of two things happens. Either supply at some terminal is exhausted, or there is some set of nodes A whose total supply is equal to the amount of flow that can leave this set in the current time bound. Recall that this is the definition of a *tight set*. The flow leaving A must be at a maximum for the remainder of the algorithm. Hence, once a set is tight, it remains tight for the remainder of the algorithm. Recall $o(A)$ is the amount of flow that can leave set A in one unit of time.

Lemma 4.2 (Meggido [9]) $o(A) + o(B) \geq o(A \cup B) + o(A \cap B)$.

A function with this property is called *submodular*. Note that any linear function satsifies this inequality at equality, so is also submodular. Since $\sigma(A)$ is linear, it is easy to see that $o(A)T - \sigma(A)$ is also submodular. For tight set A, $o(A)T - \sigma(A) = 0$ by definition. Since T is feasible if and only if $o(A)T - \sigma(A) \geq 0$, the lemma implies that the intersection of tight sets is tight and the union of tight sets is tight.

```
Quickest ($\mathcal{N}, u, \sigma, T^*$)

$T = T^*$, $i = 0$.
While $\sigma \neq 0$,
    $i = i + 1$.
    Construct two-level network with $u$, $\sigma$, $T$.
    Solve transshipment problem on this two level network.
    $f_i$ = flow through small capacity level.
    $\lambda_i$ = maximum $\lambda$ such that transshipment problem $(\mathcal{N}, u, \sigma - \lambda f_i, T - \lambda)$ is feasible.
    $d_i$ = denominator of $\lambda_i$.
    $T = d_i(T - \lambda_i)$.
    $\sigma = d_i(\sigma - \lambda_i f_i)$.
Return $\{(f_1, \lambda_1), (f_2, \lambda_2/d_1), \ldots, (f_i, \lambda_i/\prod_{j=1}^{i-1} d_j)\}$.
```

Figure 2: The quickest, integral flow rate, continuous time, transshipment algorithm

Lemma 4.3 *The algorithm searches for a new feasible integral flow at most k times.*

Proof: Each time we find a new λ by finding a new tight set, we also increase the number of nested tight sets of terminals. The old tight sets are still tight. The new tight set either intersects none of these, in which case the set taken as the union of the new and old sets is also a new tight set, or the new tight set intersects some of these. By considering the intersections and unions of the new tight set with the p existing tight sets, it is possible to find a set of at least $p + 1$ nested tight sets. Since there are at most k terminals, there can be at most k nested tight sets of terminals.

Each time we deplete a supply, this reduces the number of active terminals, and thus reduces the maximum number of nested tight sets we can have. But once a terminal is the only terminal contained in a tight set but not in the next smallest tight set, its supply will not be depleted until the algorithm ends. This is true because the larger set must send out a constant amount in every interval of time which is either strictly more or strictly less than the amount leaving next smaller set. This means that the total number of nested tight sets in the algorithm never decreases, but that the number of active terminals minus the number of nested tight sets always decreases by at least one. Hence, the total number of times the algorithm stops with a non-empty reduced problem is at most k. ∎

Theorem 4.4 *The quickest transshipment problem with zero transit times can be solved in strongly polynomial time or with $O(k^2 \log Un + k \log \Sigma)$ or $O(kn^2 \log^2 n)$ maximum flow computations.*

Proof: The correctness has been demonstrated above. Since $T^* \leq \Sigma$, the first λ falls in the range $[1, \Sigma]$, and may be fractional with denominator $d_1 \leq Um$. We then multiply remaining supplies by d_1. Repeating this, we find the k^{th} λ within $[1, d_1 d_2 \cdots d_{k-1} \Sigma]$, and it may be fractional with denominator $d_k \leq Um$. This together with Lemma 4.3 implies that the algorithm consists of at most k binary searches over range $[1, \Sigma D]$ where $D \leq d_1 d_2 \cdots d_k \leq (Um)^k$. Each guess for λ requires one maximum flow calculation. Thus, this algorithm requires $O(k^2 \log Un + k \log \Sigma)$ maximum flow computations. Using Megiddo's parametric search for the parametric maximum flow problem instead of binary search, the algorithm requires $O(kn^2 \log^2 n)$ maximum flow computations [10, 11]. ∎

4.2 Quickest discrete time integral dynamic transshipment

In this section, I explain how to convert the continuous-time integral flows into a discrete-time solution to the quickest transshipment problem. This solution will then be fully integral, and of interest for applications which can only handle integral quantities, such as train and airplane relocation problems. If all λ's are integer, then we can perform this transformation by sending f_{ij} units of flow on arc (i, j) at each of λ time units. If there are fractional λ's, then we subtract the fractional part from each λ to get a partial solution that is integral and hence satisfies an integral amount of the supplies. Since our initial supplies are integral, the sums of the supplies sent in the fractional times are also integral; and, they can be satisfied in time $\Delta = \sum_{i=1}^{k} \lambda_i - \lfloor \lambda_i \rfloor \leq k$. We can now use Observation 4.1 to solve this smaller dynamic transshipment problem with $O(k)$ maximum flow computations. These two partial solutions scheduled one after the other form a feasible, integral, discrete dynamic transshipment completing by the optimal time.

4.3 The single sink case

When there is only one sink, the dynamic transshipment problem is also known as the evacuation problem. In this section, I explain how to compute a quickest integral evacuation with $O(k)$ maximum flow computations. The best previous solution for computing a quickest integral evacuation is the algorithm of Hoppe and Tardos [8] which uses the ellipsoid method as a subroutine. The algorithm of Hajek and Ogier computes a fractional solution using $O(n)$ maximum flow computations.

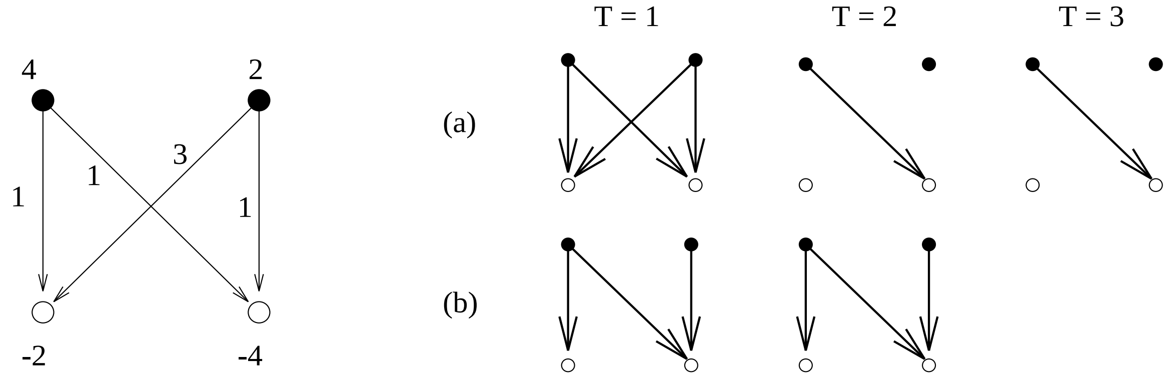
When there is only one sink (or one source), we can speed up the search for λ by using the parametric flow ideas of Gallo, Grigoriadis, and Tarjan [6], to find each λ with one maximum flow. For this case, the parametric flow problem that we solve to find λ is simpler. The new network has only one new node, a super-source—the single sink obviates the need for a super-sink. Now the arcs from super-source to source i have capacity $\sigma_i - \lambda f_i$, and all original network arcs have capacity $u_{ij}(T - \lambda)$. Gallo, et al. show that if the capacities on arcs leaving the source are non-increasing functions of a parameter, and all other capacities are constant, then the minimum value of the parameter for the source to be a minimum cut can be found in the same asymptotic time as computing one maximum flow. In our example, let $\alpha = T - \lambda$. Dividing all capacities by α gives us an equivalent problem with the original constant capacities on the original arcs, and capacities which are non-increasing functions of $1/\alpha$ on the arcs leaving the source. This is the form required by the parametric flow algorithm of Gallo, et al. Thus, for the special case of a single sink, the quickest transshipment problem—also known as the quickest evacuation problem—can be solved with at most $O(k)$ maximum flows.

Theorem 4.5 *The quickest evacuation problem with zero transit times can be solved with $O(k)$ maximum flow computations.*

5 Universally quickest dynamic transshipment

The universally quickest dynamic transshipment is a flow that minimizes the amount of excess left in the network at every moment of time. Hajek and Ogier [7] solve this problem with $O(n)$ maximum flows when there is only one sink. In this section, I describe an algorithm that solves this special case in the same asymptotic time as one maximum flow computation.

Figure 3: An example with no universally quickest transshipment



If there are multiple sinks, a universally quickest solution may not exist. For example, consider the network in Figure 3. In one unit of time it is possible to satisfy four units of demand by sending one unit of supply from each sink to each source (Fig. 3a). In two units of time it is possible to satisfy six units of demand by sending one unit from the first source to both sinks, and one from the second source to the second sink, for each of the two time units (Fig. 3b). However, if in the first time unit, all supply is sent from the second source, and all demand has been met at the first sink, then in the second time unit, it is only possible to send supply from the first source to the second sink, and the amount is restricted to one unit by the capacity of the arc. Hence if the maximum amount of supply is sent in the first time unit, it is not possible to maximize supply sent by the second unit of time.

The problem with multiple sinks is that, in the rush to send flow, some source may send flow to the wrong sink. This is not a problem when there is only one sink.

The universally quickest transshipment algorithm described here will find a series of subsets of V, $A_1 \subset A_2 \subset \cdots \subset A_r$, such that each set contains at least one more source than the previous set, and A_r contains all sources but not the sink. Each set will have a corresponding time bound T_i with $T_1 > T_2 > \ldots > T_r$, such that, in the constructed solution, the rate of flow leaving set A_i equals $o(A_i)$ from time 0 until time T_i after which all sources in $A_i \backslash A_{i-1}$ are empty.

Theorem 5.1 *If a single sink, dynamic transshipment sends flow out of node set A_i at rate $o(A_i)$ from time 0 until time T_i, after which all sources in $A_i \backslash A_{i-1}$ are empty of supply, for all A_i and T_i as defined above, then the transshipment is a universally quickest transshipment.*

Proof: By the same argument in the proof of Lemma 3.1, the maximum amount of supply that can be sent to the sink by time T equals the maximum static flow in an extended dynamic network with the capacity of each original arc multiplied by time bound T and arcs from a supersource to each source with capacity equal to the supply at the adjacent source. Represent this quantity by $s(T)$. This is also the value of a minimum cut in the extended network. A universally quickest transshipment must therefore send supply of $s(T)$ to the sink by time T, for every T. I show that a dynamic transshipment obeying the conditions of the theorem achieves this.

Proof is by induction on c, the number of sets A_i. If $c = 1$, then A_1 contains all sources, and flow leaving A_1 equals $o(A_1)$ from time 0 until time T_1. This is clearly a universally quickest evacuation.

Suppose the algorithm works correctly for $c < r$, and consider the case when $c = r$. Now consider the smallest time bound T_r found by the algorithm. This time bound corresponds to the set A_r that contains all sources. The minimum cut in the maximum flow network parameterized by T_r contains A_r. Because A_r contains all sources, only original network arcs cross this cut. Hence, this cut is also a minimum cut in the corresponding maximum flow network for every $T \leq T_r$. So the dynamic flow that sends flow out of A_r at rate $o(A_r)$ at every moment $0 \leq \theta \leq T_r$ is universally quickest flow in this time interval.

By construction, after time T_r, all sources in $A_r \backslash A_{r-1}$ are empty of supply. In addition, up until time T_r, all other sets A_i have been sending flow out at rate equal to $o(A_i)$. Finally, note that $T_i > T_r$ for all $i < r$. Thus, the algorithm run on the altered problem with no supplies at the sources in $A_r \backslash A_{r-1}$ also sends flow out of all A_i at rate $o(A_i)$ up until time T_r. By induction, this flow is a universally quickest evacuation. In particular, it is universally quickest at all times after T_r. Since, before time T_i, the flow on all arcs entering or leaving $A_i \backslash A_{i-1}$ is fixed, the flow within $A_i \backslash A_{i-1}$ can be determined independently in this time period. So any flow that maintains the appropriate flow conservation constraints at all nodes in $A_i \backslash A_{i-1}$ for $1 \leq i < r$ for $T \leq T_r$ is equivalent to any other. In particular, both flows discussed above are interchangeable on A_r for the interval $[0, T_r)$. Hence, the output of the algorithm on the original problem with r sets is also a universally quickest evacuation. ■

The algorithm has three stages: (a) It first finds sets A_i using the parametric flow algorithm of Gallo, Grigoriadis, and Tarjan [6], which also returns the corresponding time bounds T_i. These A_i are the minimum cuts in the corresponding maximum flow network parameterized by T_i. (b) This information is then used to construct a static flow which represents the initial flow rate. (c) Finally, the dynamic flow is constructed from this static flow by reducing the flow rate along source-sink paths when the supplies at the corresponding sources are depleted.

(a) Finding nested sets A_i and time bounds T_i

Consider the following parametric flow problem. Add a supersource to the original network, connnected to each original source by an arc of capacity equal to the supply at the source, divided by parameter T, the time bound. The value of a maximum flow in this network is a decreasing function of T. This parametric problem is equivalent to the problem with constant capacities on the added arcs, and T times the original capacities on the original arcs, in the sense that any maximum flow in one problem can be transformed into a feasible flow in the other by multiplying (or dividing) all flow quantities by T. This latter flow problem is the one discussed in the proof of Theorem 5.1. Gallo, Grigoriadis, and Tarjan [6] notice that the value of the maximum flow in this setting is a piecewise linear function of T with at most k breakpoints, and that the associated minimum cuts form a nested set of cuts. They describe an algorithm to find these breakpoints and cuts in $O(nm \log(n^2/m)$ time. Let the T_i be the values of T at these breakpoints and define A_i to be the minimum cut for T in interval $[T_{i+1}, T_i)$. Using the Gallo et al. algorithm, all A_i and T_i are found in time $O(nm \log(n^2/m))$, i.e. the same asymptotic time as one maximum flow computation.

(b) Constructing the static flow

Given the A_i and the T_i, we construct a static flow f on the original network with the following properties: Flow conservation holds at all non-terminal nodes, there is net flow out of all sources, and any arc that leaves any A_i is saturated in f. In addition, for any source in $A_i \backslash A_{i-1}$, the net flow in f out of the source equals the supply at the source divided by T_i. Such a static flow is useful because we can let f be the flow rate in a dynamic transshipment until time T_r, when the supplies in $A_r \backslash A_{r-1}$ are depleted. After time T_i, we can subtract path flows from f to get a new static flow which will be the flow rate until the next time bound.

We construct f in a piecewise fashion by partitioning the network into subnetworks, defining f on each subnetwork, and then defining f on all arcs that cross between subnetworks. First consider the subnetwork induced by the vertices in A_1. For each arc (x, y) in the original network that leaves this set, add its capacity to the demand at x. Include the supplies of sources in this set divided by T_1. The flow that solves the resulting transshipment problem defines f on this subgraph. In general, consider the subgraph induced by the vertices in $A_i \backslash A_{i-1}$. For each arc (x, y) in A_i that enters this set, increase the supply at y by u_{xy}. For each arc (x, y) with $x \in A_i \backslash A_{i-1}$ that leaves A_i, increase the demand at node x by u_{xy}. Also include the supplies of sources in this set divided by T_i. The flow that solves the resulting transshipment problem defines f restricted to this subgraph. On any arc leaving any A_i, f is set equal to the capacity of the arc; and f is set to 0 on any arc entering any A_i. All arcs not considered so far lie completely outside A_r. For these, consider the subgraph on nodes $V \backslash A_r$. For all arcs entering this set, increase the supply at the adjacent node by the capacity of the arc. Put demand at the sink equal to the sum of these new supplies. The flow that solves the corresponding transshipment problem defines f on this subgraph.

The bottleneck in computing f is computing the maximum flow on each subgraph. The sum of the size of the subgraphs equals the size of the original network, however, so that the time to compute f is the same asymptotic time as one maximum flow computation.

(c) Constructing the dynamic flow
Given static flow f, we compute a universally quickest transshipment by computing, for time interval $[0, T_r)$ and for each time interval of form $[T_i, T_{i-1})$, a static flow that defines the flow rate in each interval. To start, $f_r = f$ is the flow rate from time 0 until time T_r. At time T_i, the supplies of the sources in $A_i \backslash A_{i-1}$ are depleted, so f_i is reduced by the flow leaving these sources to form f_{i-1}. To compute these successive static flows, decompose f into paths and cycles. Let f_i denote the desired flow rate in interval $[T_{i+1}, T_i)$, with $f_r = f$. Once f_{i+1} is obtained, f_i can be computed by reducing flow along paths with positive flow leaving sources in $A_i \backslash A_{i-1}$.

Ford and Fulkerson explain how a flow decomposition can be computed efficiently in $O(mn)$ time [5]: Until no arc carries flow, find a simple cycle or source-sink path with flow, and subtract the maximum amount of flow possible from this path or cycle. Each subtraction reduces the flow on some arc to 0, so there are at most m subtractions. Finding a simple source-sink path or cycle takes at most $O(n)$ time, as does subtracting the flow.

Note that all three steps of the algorithm run in time asymptotic to one maximum flow computation. Thus the run time of the algorithm is established.

Theorem 5.2 *The single sink, universally quickest transshipment problem can be solved in the same asymptotic time as one maximum flow computation.*

6 Incoming and outgoing traffic

While the solving a transshipment problem with fixed supplies is useful for clearing a network after a communication breakdown, every day usage more often involves continuous streams of traffic. The algorithms presented here, like the algorithm of Hajek and Ogier [7], allow for constant streams of flow into or out of any node in the network.

If there is constant stream of flow into or out of any nodes, the sum of the rates of these flow must equal zero in order for the problem to remain stable. Before solving the dynamic transshipment part of the problem, we can determine course of this flow with one maximum flow computation. Introduce a super-source connected to all nodes with incoming flow by arcs with capacity equal to the rate of the incoming flow for that node. Similarly, introduce a super-sink, and connect all nodes with outgoing flow to the super-sink with arcs of capacity equal to the rate of outgoing flow. If the maximum flow has value strictly less than the sum of the rates of incoming flow, then excess will build up in the network, and the problem is infeasible. Otherwise, the maximum flow determines the course these external flows will take through the network. The residual network of this flow—the network of arcs e with $u_e - f_e > 0$—is passed on to the main algorithm.

7 Acknowledgements

I would like to thank Éva Tardos for helpful comments on various drafts of this paper, and for pointing out the much simpler lower bound in Section 4.2.

References

[1] E. J. Anderson and P. Nash. *Linear Programming in Infinite-Dimensional Spaces.* John Wiley & Sons, 1987.

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

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

[4] L. Fleischer and E. Tardos. Efficient continuous time dynamic network flow algorithms. Technical Report TR1166, Cornell University, Department of Operations Research and Industrial Engineering, 1996.

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

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

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

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

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

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

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

[12] A. B. Philpott. Continuous-time flows in networks. *Mathematics of Operations Research*, 15(4):640–661, November 1990.

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

[14] M. C. Pullan. An algorithm for a class of continuous linear programs. *SIAM J. Control and Optimization*, 31(6):1558–1577, November 1993.