Chapter 1
Polynomial Time Algorithms for Some Evacuation Problems*

Bruce Hoppe† Éva Tardos‡

Abstract

Evacuation problems can be modeled as flow problems on dynamic networks. A dynamic network is defined by a graph with capacities and integral transit times on its edges. The maximum dynamic flow problem is to send a maximum amount of flow from a source to a sink within a given time bound T; conversely, the quickest flow problem is to send a given flow amount v from the source to the sink in the shortest possible time. These dynamic flow problems have been studied previously and can be solved via simple minimum cost flow computations.

More complicated dynamic flow problems have numerous applications and have been studied extensively. There are no polynomial time algorithms known for many of these problems, including the quickest flow problem with just two sources, each with a flow amount that must reach a single sink. The general multiple source quickest flow problem is commonly used as a model for building evacuation; we also call it the evacuation problem.

In this paper we consider three problems related to the evacuation problem. We give a polynomial time algorithm for the evacuation problem with a fixed number of sources and sinks. We give a polynomial time algorithm for the lexicographic maximum dynamic flow problem with any number of sources; in this problem we seek a dynamic flow that lexicographically maximizes the amounts of flow leaving the sources in a specified order. Our algorithm for the evacuation problem follows as an application.

We also consider the earliest arrival flow problem. Given a source, sink, and time bound T, an earliest arrival flow maximizes the amount of flow reaching the sink at every time step up to and including T. The existence of such a flow is well known, but there are no polynomial time algorithms known even to approximate it. We give a polynomial time algorithm that for any fixed $\epsilon > 0$ approximates an earliest arrival flow within a factor of $1+\epsilon$.

*Tecnical Report ORIE-1117 of the School of Operations Research and Industial Engineering, Cornell University. Research was done while the authors were visiting the Department of Computer Science at Princeton University.

†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, an NSF PYI award, and by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through NSF grant DMS-8920550.

1 Introduction

When a bomb explodes beneath a skyscraper, the occupants must evacuate the building as quickly as possible. Panicked stampeding through hallways and down stairwells can easily lead to catastrophic bottlenecks. With hopes of preventing such disasters, engineers devise evacuation plans; in an emergency, the occupants need only follow the appropriate plan in order to exit the building safely.

Dynamic networks can model building evacuation and many other problems as well. A dynamic network is defined by a directed graph $G = (V, E)$ with sources, sinks, and non-negative capacities u_{xy} and non-negative integral transit times τ_{xy} for every edge $xy \in E$. In a feasible dynamic flow, at most u_{xy} units of flow can be pipelined along edge xy with each time step; flow leaving x at time θ reaches y at time $\theta + \tau_{xy}$.

Two classical dynamic network flow problems are the maximum dynamic flow problem and the quickest flow problem; each applies to networks with only one source and one sink. A maximum dynamic flow sends as much flow as possible from the source to the sink within a specified time bound T. Conversely, a quickest flow sends a specified amount v of flow from the source to the sink in the shortest possible time T. Ford and Fulkerson [4] showed that the maximum dynamic flow problem can be solved via a single minimum-cost flow computation. The quickest flow problem can be reduced to the maximum dynamic flow problem by binary search; Burkard, Dlaska and Klinz [2] gave more efficient and strongly polynomial algorithms for this problem.

The quickest flow problem with multiple sources, each with a flow amount that must reach the sink, is commonly used to model building evacuation; we also call it the evacuation problem. Dynamic network flow problems with several sources and sinks also arise in many other applications (*e.g.*, airline, truck, and railway scheduling). While there has been a fair amount of work in this area (see the surveys [1, 13] and the bibliography), there are no polynomial time algorithms known for most of these problems, including

the evacuation problem with just two sources. We give a polynomial time algorithm for the evacuation problem with a fixed number of sources and sinks.

It is not obvious how to describe a dynamic flow efficiently when the time bound is large. All previous polynomial time dynamic network flow algorithms have relied on the notion of a temporally repeated flow to describe a dynamic flow. Ford and Fulkerson [4] introduced temporally repeated flows — a very simple class of dynamic flows defined in Section 3 — and proved that the maximum dynamic flow problem always has a temporally repeated solution. This is not the case for many simple variations of the problem, but no efficient alternative to temporally repeated flows has been proposed. In this paper, we describe a generalization of temporally repeated flows, and we consider a number of dynamic network flow problems with generalized temporally repeated solutions.

Our first result relates to the evacuation problem with a single source. Suppose a bomb explodes beneath a skyscraper on Sunday, when the only occupants are tourists on the observation deck. The quickest flow problem models a plan to minimize the time required for all the tourists to leave the building. Given the increasing amounts of fire and smoke, however, we not only want the last person to evacuate as soon as possible, we also would like that at every intermediate time there are as few people in the building as possible. This extra restriction is modeled by the earliest arrival flow problem.

More formally, an earliest arrival flow simultaneously maximizes the amount of flow reaching the sink at every time step $\theta \leq T$. The existence of such a flow was proved by Gale [6]; and exponential algorithms to compute it were described independently by Wilkinson [14] and Minieka [11]; but there is no polynomial time algorithm known even to approximate this flow. We give a polynomial time algorithm that for any fixed $\epsilon > 0$ finds a $(1+\epsilon)$-approximate earliest arrival flow: at every time step $\theta \leq T$ the amount of flow that has reached the sink is within a factor of $1+\epsilon$ of the maximum possible. Our algorithm is a capacity scaling shortest augmenting path algorithm; however, it has the unusual feature of scaling upwards. First we use all edges for augmentation; after sufficient augmentations, the algorithm periodically rounds down the remaining capacity and restricts all further augmentations to the rounded network.

We next consider the lexicographic maximum dynamic flow problem. In this problem we are given an arbitrary number of sources; the sources are ordered to distinguish high priority sources from low priority ones. Given a time bound T, we seek to maximize the amounts that the sources send to the sink in the given lexicographic order. We give a polynomial time algorithm for this problem. Interestingly, our solution is analogous to an infinite horizon temporally repeated flow — one with no time bound at all; however, because the flow is identically zero after time T, it is equivalent to a flow with time bound T.

Finally, we consider the evacuation problem with a fixed number of sources and sinks. Klinz [9] observed that the minimum time required for evacuation with a fixed number of sources can be computed in polynomial time. However, her algorithm finds only the required time, and not the evacuation plan. We extend her results by giving a polynomial time algorithm to find the optimal flow for any fixed number of sources. The solution to the evacuation problem with k sources is obtained by combining the solutions to the $k!$ lexicographic maximum dynamic flow problems, one for each ordering of the sources. The algorithm can be extended to solve the evacuation problem with a fixed number of sources and sinks.

2 Definitions

A *dynamic network* $\mathcal{N} = (G, u, \tau, \mathcal{S}, \mathcal{T})$ consists of a directed graph $G = (V, E)$ with a non-negative *capacity* u_{xy} and non-negative integral *transit time* τ_{xy} associated with each edge $xy \in E$, and a set of sources $\mathcal{S} \subseteq V$ and sinks $\mathcal{T} \subseteq V$. We will also refer to transit times as *length*. We use n and m to denote the number of vertices and edges in G. For notational simplicity we assume that there are no parallel or opposite edges and no zero length cycles in G. We use $u_{xy} = 0$ when $xy \notin E$, and $\tau_{yx} = -\tau_{xy}$ for all $xy \in E$; we will also refer to the opposite pairs yx as edges. We assume that sources have no entering edges in E, and sinks have no leaving edges in E. We consider first the case with a single source s and a single sink t.

A *dynamic (s,t)-flow* is a function $f_{xy}(\theta)$ on $E \times \mathbb{N}$ that satisfies the conservation constraints

$$\forall x \in V \setminus \{s,t\},\ \theta \in \mathbb{N}: \quad \sum_{y \in V} f_{xy}(\theta) \ = \ 0,$$

where we use the notation that $f_{yx}(\theta + \tau_{xy}) = -f_{xy}(\theta)$ for every edge $xy \in E$ and time θ, $f_{xy}(\theta) = 0$ for all θ if neither xy nor yx is in E, and $f_{xy}(\theta) = 0$ for all xy if $\theta < 0$. Dynamic flow f is *feasible* if it satisfies capacity constraints $f_{xy}(\theta) \leq u_{xy}$ for every edge xy and time θ. Notice that by the symmetric notation for an edge $xy \in E$, the inequality $f_{yx}(\theta + \tau_{xy}) \leq u_{yx}$ requires that $f_{xy}(\theta) \geq 0$. The *dynamic value* of f given time bound $T \in \mathbb{N}$ is the net dynamic flow into t for all time up to

T, defined as

$$|f|_T = -\sum_{\theta=0}^{T} \sum_{x \in V} f_{tx}(\theta).$$

In the absence of any time bound, $|f|$ is the analogous infinite sum. If a dynamic flow f is only defined up to some time bound $T \in \mathbb{N}$ (or equivalently it is zero for all time after T) then f is a *finite horizon* dynamic flow; otherwise it is an *infinite horizon* flow. Dynamic flows with many sources and sinks are defined analogously.

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 $|f|_T$ 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 $|f|_T \geq v$. In the *earliest arrival flow problem* we seek a feasible dynamic flow f with specified time horizon T that simultaneously maximizes the amount of flow reaching the sink at every time step $\theta \leq T$. Such flows are also known in the literature as *universal maximal dynamic flows*.

In the *lexicographic maximum dynamic flow problem* there are k sources $\mathcal{S} = \{s_1, \ldots, s_k\}$ and we seek a feasible dynamic flow with specified time horizon T that lexicographically maximizes the amounts leaving the sources. The *evacuation problem* is the multiple source version of the quickest flow problem. We are given amounts $v_s \geq 0$ for every source $s \in \mathcal{S}$, and the objective is to minimize the time T so that there is a feasible dynamic flow f with time horizon T where the amount of flow leaving source s is at least v_s.

We will refer to flows in G in the traditional sense as static flows. A *static (s,t)-flow* is a function g on E that satisfies the conservation constraints $\sum_y g_{xy} = 0$ for every node $x \neq s,t$, where we use again the symmetric notation that $g_{yx} = -g_{xy}$ for every edge $xy \in E$, and $g_{xy} = 0$ if neither xy nor yx is in E. Static flow g is *feasible* if it also satisfies capacity constraints $g_{xy} \leq u_{xy}$ for every xy. The *residual graph* of the static flow g subject to capacities u is defined as $G_{u,g} = (V, E_{u,g})$, where $E_{u,g} = \{xy : g_{xy} < u_{xy}\}$; the associated *residual capacity* function is $u^g_{xy} = u_{xy} - g_{xy}$. We will also use the notation G_g and E_g when u is clear. An edge with zero residual capacity is *saturated*. Note that due to the symmetric notation for an edge $xy \in E$, the opposite edge yx is saturated if $g_{xy} = 0$. The *static value* of g is $|g| = \sum_x g_{xt}$. Static flows with many sources and sinks 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 $x \in V$ has $T+1$ copies in $V(T)$, denoted $x(0), \ldots, x(T)$. Each edge $xy \in E$ has $T - \tau_{xy} + 1$ copies in $E(T)$, denoted $x(\theta)y(\theta + \tau_{xy})$ for any time $0 \leq \theta \leq T - \tau_{xy}$. In addition, $E(T)$ contains a *holdover* edge $x(\theta)x(\theta+1)$ with infinite capacity for each vertex x and time $\theta < T$. Dynamic flow f is a static flow in $G(T)$ where the value of the flow on edge $x(\theta)y(\theta + \tau_{xy})$ is $f_{xy}(\theta)$. 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. An infinite horizon dynamic flow is equivalent to a static flow in the infinite time-expanded graph $G(*)$, defined analogously to $G(T)$.

3 Generalized Temporally Repeated Flows

If the time bound T is large, then it is not clear how to describe a dynamic flow in polynomial time. Ford and Fulkerson [4] introduced temporally repeated flows to represent some dynamic flows. All previously known polynomial time dynamic network flow algorithms use this representation, which we define below.

A *chain flow* $\gamma = \langle v, P \rangle$ is a static flow of value v along the path P. Let $\tau(\gamma)$ denote the length of P. Let $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ be a multiset of chain flows. We say that Γ is a *chain decomposition* of static flow g if $\sum_{i=1}^{k} \gamma_i = g$; and furthermore, Γ is a *standard chain decomposition* of g if all of its chain flows use edges in the same direction as g does. We say that γ is a chain flow "in" g if there is some standard chain decomposition of g that contains γ.

A standard chain decomposition Γ of a feasible static flow g can be used to induce a feasible dynamic flow. Given time bound T, suppose that every chain flow in Γ has length at most T. Each chain flow $\gamma = \langle v, P \rangle \in \Gamma$ induces a finite horizon dynamic flow simply by sending v units of flow every time step from time zero till time $T - \tau(\gamma)$. Summing these induced dynamic flows for all $\gamma \in \Gamma$ yields a feasible dynamic flow that ends at time T. Such a dynamic flow is called a *temporally repeated flow*, and we denote it by $[\Gamma]^T$. Analogously, we denote by $[\Gamma]$ the dynamic flow induced by repeating each chain flow endlessly. Note that $|[\Gamma]^T|_\theta = |[\Gamma]|_\theta$ for every $\theta \leq T$.

Ford and Fulkerson observed that the dynamic value of $[\Gamma]^T$ depends only on the static flow g, and is independent of the choice of the standard chain

decomposition Γ. The dynamic flow value can be expressed as

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

$$(2) \qquad = (T+1)|g| - \sum_{xy \in E} \tau_{xy} g_{xy}.$$

This implies that finding a maximum temporally repeated dynamic flow is equivalent to a minimum cost circulation problem: assign every edge xy cost $c_{xy} = \tau_{xy}$ 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 with one minimum cost circulation computation.

Unfortunately, many dynamic network problems do not yield to this efficient approach. Neither the earliest arrival flow problem, nor the lexicographic maximum dynamic flow problem, nor the evacuation problem necessarily have temporally repeated solutions.

We use non-standard chain decompositions to induce dynamic flows: the chain flows in the decomposition may use oppositely directed flows on arcs. We refer to the resulting dynamic flows as *generalized temporally repeated flows*. Notice that equation (2) is valid on any chain decomposition; however, when using a non-standard chain decomposition, the resulting dynamic flow might not be feasible. Consider a feasible chain flow γ that uses an edge $xy \in E$ and a chain flow γ' that uses the opposite edge yx; then γ' "cancels" the flow of γ on xy. If the portion of γ from the source to x is longer than the corresponding part of γ', then the resulting dynamic flow is not feasible: the flow γ' reaches yx too early — before the flow γ (which it needs to cancel) has reached xy. A similar problem occurs if the portion of γ from y to the sink is longer than the corresponding part of γ': the flow that γ' needs to cancel has left already.

The earliest arrival flow algorithms of Wilkinson [14] and Minieka [11] can be viewed as computing a generalized temporally repeated flow. Suppose that a static minimum cost maximum flow g^* in G with costs $c_{xy} = \tau_{xy}$ is computed via the shortest augmenting path algorithm of Ford and Fulkerson [4]. The augmentations of this algorithm define a chain decomposition Γ^* of g^*. Wilkinson and Minieka showed that $[\Gamma^*]^T$ is an earliest arrival flow. We will refer to the chain decomposition Γ^* as a *chain decomposition inducing an earliest arrival flow*.

The proof that $[\Gamma^*]^T$ is a feasible dynamic flow depends on the property of the shortest augmenting path algorithm: for any vertex $x \in V$, the shortest residual path lengths from s to x and from x to t do not decrease during the algorithm.

Theorem 3.1 $[\Gamma^*]^T$ is a feasible dynamic flow.

The proof that $[\Gamma^*]^T$ is an earliest arrival flow depends on another property of the shortest augmenting path algorithm: after a length θ augmentation, the resulting circulation is of minimum cost if the return arc ts is given cost $-\theta$. Therefore, for any time bound $\theta \le T$, the subset $\Gamma^*_\theta = \{\gamma \in \Gamma^* : \ \tau(\gamma) \le \theta\}$ induces a maximum dynamic flow with time θ. Note also that $\Gamma^* - \Gamma^*_\theta$ has no effect on the flow value until after time θ.

Theorem 3.2 $[\Gamma^*]^T$ is an earliest arrival flow.

As shown by Zadeh [15], the shortest augmenting path algorithm is not polynomial. In the next section, however, we describe a very similar algorithm that computes an approximate earliest arrival flow in polynomial time.

4 Approximate Earliest Arrival Flows

In this section we develop a capacity-scaling algorithm that computes an approximate earliest arrival flow in polynomial time. For this algorithm we assume that the capacities are integral and bounded by U. The algorithm has the unusual feature of scaling upwards. Traditional scaling algorithms work initially with capacities rounded by a big scaling factor; the idea is that large capacity edges are more important than small capacity edges. In a dynamic flow, however, a small capacity edge that is short might carry more flow than a large capacity edge that is long.

The algorithm is shown in Figure 1. In essence, it computes a minimum cost flow in an appropriately rounded static network via repeated shortest augmenting paths. We use the chain decomposition defined by the sequence of augmentations to induce a dynamic flow. The rounding guarantees that the number of augmentations can be bounded by a polynomial in n, $\log U$, and ϵ^{-1}, where $\epsilon > 0$ is an error parameter.

The algorithm starts by augmenting along exact shortest paths. Depending on ϵ, the algorithm periodically rounds down the residual capacities by an increasing scaling factor Δ. This implies that all residual capacities are integer multiples of Δ, so that subsequent augmentations carry at least Δ units of flow in the static network. Notice that if an edge $xy \in E$ carries

```
Γ ← ∅;  Δ ← 1;  ũ ≡ u;  g ≡ 0
while (∃ (s,t)-path in G_{ũ,g} of length ≤ T) {
    σ ← 0
    while ( (σ < mΔ/ε) and
            (∃ (s,t)-path in G_{ũ,g} of length ≤ T) ) {
        P ← shortest (s,t)-path in G_{ũ,g}
        v ← minimum residual capacity of P
        augment g by v along P
        Γ ← Γ + {⟨v, P⟩}
        σ ← σ + v
    }
    Δ ← 2Δ
    ∀x, y ∈ V :  ũ_xy ← ũ_xy − (ũ^g_xy mod Δ)
}
```

Figure 1: Earliest Arrival Flow $(1+\epsilon)$-Approximation Algorithm

less than Δ units of flow, then rounding the residual capacity of yx results in a negative capacity for edge yx, that is, a lower bound on the flow of edge xy. This corresponds to "irreversible" flow on edge xy.

Theorem 4.1 $[\Gamma]^T$ is a feasible dynamic flow.

Proof: The rounded capacity function $\tilde{u}$ never increases on any edge throughout the algorithm. Thus, feasibility with rounded capacity $\tilde{u}$ implies feasibility with capacity u, and the monotone residual path length property used by Theorem 3.1 is maintained as well. The feasibility of $[\Gamma]^T$ then follows as in Theorem 3.1. □

Each rounding reduces the capacities, and so the maximum dynamic flow value in the rounded network might decrease. To analyze the amount of lost dynamic flow we need some additional notation. Say there are $k+1$ scaling phases during the algorithm, numbered 0 to k. We index phases so that $\Delta = 2^i$ during the inner loop of phase i. Let Γ_i denote the multiset of chain flows at the end of phase i; let T_i denote the length of the longest chain flow in Γ_i; let g_i denote the static flow after phase i; and let $\tilde{u}_i$ denote the rounded capacity function at the *beginning* of phase i.

First we analyze the decrease in the dynamic flow value due to a single rounding. Let Λ_i^* denote a chain decomposition inducing an earliest arrival flow in the residual graph $G_{\tilde{u}_i,g_i}$, and let $\tilde{\Lambda}_i$ denote a chain decomposition inducing an earliest arrival flow in the further rounded residual graph $G_{\tilde{u}_{i+1},g_i}$. For any time θ, the loss in dynamic flow value due to the rounding at the end of phase i is $|[\Lambda_i^*]|_\theta - |[\tilde{\Lambda}_i]|_\theta$. The following lemma bounds this loss by ϵ times the value of the dynamic flow induced by the chain flows added during phase i:

Lemma 4.2 $|[\Lambda_i^*]|_\theta - |[\tilde{\Lambda}_i]|_\theta \;\le\; \epsilon|[\Gamma_i - \Gamma_{i-1}]|_\theta$

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

$$|[\Lambda_i^*]|_\theta - |[\hat{\Lambda}_i]|_\theta \;\le\; m2^i(\theta - T_i).$$

Note that $[\hat{\Lambda}_i]^T$ is feasible for the earliest arrival flow problem defined on the rounded residual graph $G_{\tilde{u}_{i+1},g_i}$, but $[\tilde{\Lambda}_i]^T$ is feasible and optimal for the same problem, so that $|[\tilde{\Lambda}_i]|_\theta \ge |[\hat{\Lambda}_i]|_\theta$, and therefore

$$|[\Lambda_i^*]|_\theta - |[\tilde{\Lambda}_i]|_\theta \;\le\; m2^i(\theta - T_i).$$

Finally, since the chain flows of $\Gamma_i - \Gamma_{i-1}$ have total value at least $m2^i/\epsilon$, and every chain flow has length at most T_i, we obtain

$$|[\Gamma_i - \Gamma_{i-1}]|_\theta \;\ge\; (m2^i/\epsilon)(\theta - T_i).$$

□

Theorem 4.3 For any time $\theta \le T$, let v_θ^* denote the maximum dynamic flow value in time θ. The algorithm of Figure 1 computes dynamic flow $[\Gamma]^T$ in time $O(m\epsilon^{-1}(m + n\log n)\log U)$ such that $v_\theta^* \le (1+\epsilon)|[\Gamma]^T|_\theta$.

Proof: The claimed running time follows easily. There there are $O(\log U)$ scaling phases. Capacity rounding guarantees that there are $O(m/\epsilon)$ augmentations per phase, each of which requires one non-negative edge length shortest path computation of complexity $S(m,n)$, where $S(m,n) = O(m + n\log n)$ [5].

We use Lemma 4.2 to prove the approximate optimality. Theorem 3.2 implies that phase $i+1$ begins to compute an earliest arrival flow in the rounded residual graph $G_{\tilde{u}_{i+1},g_i}$ of phase i. This means that

$$|[(\Gamma_{i+1} - \Gamma_i) + \Lambda_{i+1}^*]|_\theta \;=\; |[\tilde{\Lambda}_i]|_\theta.$$

Similarly, we have $v^*_\theta = |[\Gamma_0 + \Lambda^*_0]^T|_\theta = |[\Gamma_0 + \Lambda^*_0]|_\theta$. Now we get the following chain of equalities:

$$\begin{aligned}
&|[\Gamma_0]|_\theta + |[\Lambda^*_0]|_\theta \\
&\quad = \quad |[\Gamma_k]|_\theta + \sum_{i=0}^{k-1} \Big(|[\Gamma_i]|_\theta - |[\Gamma_{i+1}]|_\theta \Big) + |[\Lambda^*_0]|_\theta \\
&\quad = \quad |[\Gamma_k]|_\theta + \sum_{i=0}^{k-1} \Big(|[\Lambda^*_{i+1}]|_\theta - |[\tilde{\Lambda}_i]|_\theta \Big) + |[\Lambda^*_0]|_\theta \\
&\quad = \quad |[\Gamma_k]|_\theta + \sum_{i=0}^{k-1} \Big(|[\Lambda^*_i]|_\theta - |[\tilde{\Lambda}_i]|_\theta \Big) + |[\Lambda^*_k]|_\theta .
\end{aligned}$$

Since the phase-k residual graph $G_{\hat{u}_k, g_k}$ contains no (s,t)-paths of length less than θ, it follows that $|[\Lambda^*_k]|_\theta = 0$. Applying Lemma 4.2 and observing $|[\Gamma_{k-1}]|_\theta \le |[\Gamma_k]|_\theta$, we obtain:

$$|[\Gamma_0 + \Lambda^*_0]|_\theta \;\le\; |[\Gamma_k]|_\theta + \epsilon |[\Gamma_k]|_\theta .$$

□

5 Lexicographic Maximum Dynamic Flows

Minieka [11] and Megiddo [10] showed that in a static network with source set $\mathcal{S} = \{s_1, \ldots, s_k\}$ there exists a feasible flow such that the amount of flow leaving the sets of sources $\mathcal{S}_i = \{s_1, \ldots, s_i\}$ is simultaneously maximum for every i. Such a flow is clearly a lexicographically maximum flow. This observation applies to the formulation of the lexicographic maximum dynamic flow problem using the time-expanded graph.

Figure 2 presents a polynomial time algorithm that computes chain decomposition Γ such that the dynamic flow $[\Gamma]$ is a lexicographic maximum dynamic flow. The chain decomposition seems to induce an infinite horizon flow; however, there is no flow left in the network after time T.

We introduce a supersource s, incident to infinite capacity artificial edges ts with transit time $-(T+1)$ and ss_i with zero transit time for all $1 \le i \le k$. Let G^k denote the resulting graph. First we compute a minimum cost circulation g^k in the static network using transit times as costs. The rest of the algorithm consists of k iterations indexed in descending fashion. In iteration i 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 to the flow g^{i+1} to obtain flow g^i. Each flow f^i yields a standard chain decomposition. The assumption that

---

$V \leftarrow V \cup \{s\}$
$E \leftarrow E \cup \{ts\} \cup \{ss_i : i = 1, \ldots, k\}$
  where $u_{ts} = u_{ss_i} = \infty$, $\tau_{ts} = -(T+1)$, $\tau_{ss_i} = 0$
$g \leftarrow$ minimum cost circulation using τ as edge costs
$\Gamma \leftarrow$ standard chain decomposition of g
for $i \leftarrow k-1, \ldots, 0$ {
  delete edge ss_{i+1} from E
  $f \leftarrow$ minimum cost maximum (s, s_{i+1})-flow in G_g
    using τ as edge costs
  $g \leftarrow g + f$
  $\Delta \leftarrow$ standard chain decomposition of f
  $\Gamma \leftarrow \Gamma + \Delta$
}

Figure 2: Lexicographic Maximum Dynamic Flow Algorithm

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 the arc st leave t at time $T+1$. We use $p^i(x)$ to denote the shortest residual path length from s to vertex $x \in V$ in $G^i_{g^i}$; Γ^i denotes the chain decomposition accumulated by iteration i; and $\Delta^{i-1} = \Gamma^{i-1} - \Gamma^i$.

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

Artificial edge ts and the assumption that s_{i+1} has no entering edges in G guarantee that after iteration i, the node s_{i+1} is balanced in the static flow g^i. This implies the following lemma:

Lemma 5.1 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 observation:

Lemma 5.2 For any vertex $x \in V$ and any iteration i, $p^i(x) \ge p^{i+1}(x)$.

Lemma 5.2 follows from the facts that that every iteration computes a minimum cost flow, and we delete and do not add edges between iterations. To prove the correctness of the algorithm we also need the following lemmas:

Lemma 5.3 Suppose $\gamma \in \Delta^{i-1}$ and $x, y \in V$. Then γ does not cover xy at any time before $p^i(x)$.

Proof: By definition, the residual graph $G^i_{g^i}$ contains no (s,x)-path of length less than $p^i(x)$. The path of any chain flow in Δ^{i-1} is a path in $G^i_{g^i}$, and hence cannot cover xy at any time before $p^i(x)$. □

Lemma 5.4 Suppose $\gamma \in \Delta^i$ and $x, y \in V$. If γ touches xy, then γ covers xy at time $p^i(x)$.

Proof: By contradiction. If γ touches xy, but does not cover it at time $p^i(x)$, then the path of γ gives a residual path from x to s in $G^i_{g^i}$ of cost less than $-p^i(x)$. Together with the definition of $p^i(x)$ this means that the residual graph $G^i_{g^i}$ has a negative cycle, contradicting Lemma 5.1. □

Theorem 5.5 $[\Gamma]$ is a feasible dynamic flow.

Proof: We prove that $[\Gamma^i]$ obeys all capacity constraints by reverse induction on i. Γ^k is a standard chain decomposition of g^k, therefore $[\Gamma^k]$ obeys all capacity constraints. Suppose $1 \le i \le k$, and $[\Gamma^i]$ is feasible; we show that $[\Gamma^{i-1}]$ is feasible. Note that $[\Gamma^{i-1}] = [\Gamma^i] + [\Delta^{i-1}]$. Consider any edge xy and any time θ. If $[\Delta^{i-1}]_{xy}(\theta) = 0$, then $[\Gamma^{i-1}]_{xy}(\theta) = [\Gamma^i]_{xy}(\theta)$, and so by the induction hypothesis the capacity constraint is not violated. If $[\Delta^{i-1}]_{xy}(\theta) \neq 0$, then Lemmas 5.2, 5.3, and 5.4 imply that $[\Gamma^i]_{xy}(\theta) = g^i_{xy}$. Since $[\Delta^{i-1}]$ is a feasible dynamic flow in $G^i_{g^i}$, the capacity constraint is not violated.

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

Theorem 5.6 $[\Gamma]$ is a lexicographic maximum dynamic flow.

Proof: Our proof relies on the infinite time-expanded graph $G(*)$. Given any index i : $1 \le i \le k$, we construct a $(\mathcal{S}_i, t)$ cut C_i in the time-expanded graph, and show that $[\Gamma]$ saturates C_i. More precisely, cut C_i separates the source set $\{s_j(0) : j = 1, \ldots, i\}$ from the sink node $t(T)$.

Let $C_i = \{x(\theta) : \theta \ge p^i(x)\}$. First note that for any source s_j with $j \le i$ we have $p^i(s_j) = 0$ and so $s_j(0) \in C_i$. Second, observe that $p^i(t) = T + 1$ for every i. This means $t(\theta) \in C_i$ for any time $\theta > T$ and $t(\theta) \notin C_i$ for any time $\theta \le T$. Thus, no flow from $\mathcal{S}_i$ can reach sink t by time T without crossing cut C_i; and furthermore, the net flow crossing C_i must reach sink t by time T.

Consider any non-holdover edge $x(\theta)y(\theta + \tau_{xy})$ that crosses C_i: then $x(\theta) \in C_i$ and $y(\theta + \tau_{xy}) \notin C_i$. (The other case is considered as opposite edge $y(\theta - \tau_{yx})x(\theta)$.) By the definition of C_i, this means $\theta \ge p^i(x)$ and $p^i(y) > \theta + \tau_{xy}$. Added together, these two inequalities imply that $p^i(y) > p^i(x) + \tau_{xy}$, which means that xy must be saturated by static flow g^i.

Regarding the dynamic flow, we consider the implication of the above inequalities separately. By Lemmas 5.2 and 5.4, $\theta \ge p^i(x)$ implies that any chain in Γ^i that touches xy must also cover xy at time θ. By Lemmas 5.2 and 5.3, $p^i(y) > \theta + \tau_{xy}$ implies that no chain in $\Gamma - \Gamma^i$ covers xy at time θ. Since Γ^i is a chain decomposition of g^i, these observations imply that $[\Gamma]_{xy}(\theta) = g^i_{xy}$, and hence $x(\theta)y(\theta + \tau_{xy})$ is saturated.

Finally, consider any reverse holdover edge $x(\theta + 1)x(\theta)$ that crosses C_i. (By the definition of C_i, the holdover edge cannot cross C_i.) If x is neither a source nor the sink, then flow conservation implies that $[\Gamma]$ makes no use of any holdover edge $x(\theta)x(\theta + 1)$, and so the reverse edge is saturated. If $x = t$, then $\theta = T$ and the holdover edge $x(\theta)x(\theta + 1)$ has no flow. If x is a source, there are two cases: (1) $x \in \mathcal{S}_i$. No reverse holdover edge for x crosses C_i. (2) $x \notin \mathcal{S}_i$. There is no net static flow out of x in g^i. Note also that $p^i(x) = \theta + 1$. Since no chain flow in $\Gamma - \Gamma^i$ starts in $\mathcal{S} \backslash \mathcal{S}_i$, Lemmas 5.2 and 5.4 imply that no net flow leaves x at any time after θ. This means that the holdover edge $x(\theta)x(\theta + 1)$ has no flow. □

Dynamic flow $[\Gamma]$ appears to be an infinite horizon dynamic flow, and so in Theorem 5.6 we explicitly ignore any flow in $[\Gamma]$ reaching sink t after time T; however, $[\Gamma]$ is actually a finite horizon dynamic flow in G with time bound T.

Theorem 5.7 $[\Gamma]$ has time horizon T.

Proof: For any $x, y \in V$ and any time $\theta \ge T + 1$, we claim that $[\Gamma]_{xy}(\theta) = 0$. Suppose some chain flow in Γ touches xy (otherwise the claim is trivially true). Notice that Lemma 5.1 and the assumption that G has no zero length cycles imply that the static flow g^0 is zero. By Lemmas 5.2 and 5.4, this implies that $[\Gamma]_{xy}(\theta) = 0$ for any $\theta \ge p^i(x)$, where $i + 1$ is the maximum index with $g^{i+1}_{xy} \neq 0$. Artificial edge ts guarantees that $p^i(x)$ is finite, and so there is no flow left in the network after a sufficiently large time T'. Similarly, there is no flow entering sink t after time T (because of the canceling effect of chain flows using arc st). Assume by contradiction that there is flow left somewhere in the network after time T. All edges with

positive capacity have non-negative length, so that this flow cannot reach sink t by time T, and hence must stay in the network forever. This contradicts the fact that there is no flow left in the network after time T'. □

The complexity of this algorithm is dominated by the k minimum cost flow computations, each of complexity $M(m,n)$, where $M(m,n) = O((m \log n)(m + n \log n))$ [12].

Minieka [11] and Megiddo [10] observed that a static lexicographic maximum flow problem with many sources $\mathcal{S}$ and many sinks $\mathcal{T}$ can be solved by separately solving two lexicographic maximum flow problems, one with a single supersink and many sources, and one with a supersource and many sinks. The two flows can be pasted together along a minimum $(\mathcal{S}, \mathcal{T})$ cut. This observation implies that we can extend our result to lexicographic maximum dynamic flow problems with multiple sinks in addition to multiple sources: the dynamic flow obtained is lexicographically maximum simultaneously with respect to the priorities on the sources and the sinks.

Theorem 5.8 A lexicographic maximum dynamic flow with k sources and sinks can be computed in $O(kM(m,n))$ time.

6 Evacuation with a Fixed Number of Sources and Sinks

In this section we consider the quickest flow problem with a fixed number of sources and sinks. We refer to a source or sink as a *terminal*, and let $\mathcal{D}$ denote the set of terminals. For a terminal x, let v_x denote the amount of commodity that needs to get out of x (*i.e.*, v_x is negative if x is a sink). The problem is to find a feasible dynamic flow that satisfies the demands at the sources and sinks within the shortest possible time T. If T is known, this problem is equivalent to a transportation problem in the time expanded graph $G(T)$. By the max-flow min-cut theorem of Ford and Fulkerson [4] for static networks, a feasible flow exists if for every subset X of terminals, there is a flow of value $v_X = \sum_{x \in X} v_x$ from X to $\mathcal{D} \setminus X$ in $G(T)$.

Klinz [9] observed that for any subset X of terminals, the problem of determining the minimum time T_X such that v_X total units of flow can be sent from any terminals in X to any terminals in $\mathcal{D} \setminus X$ is equivalent to a single source and single sink quickest flow problem; and the minimum time required for the quickest flow problem with k terminals is the maximum of the times T_X over the 2^k subsets $X \subseteq \mathcal{D}$. This implies that the optimal time for the quickest flow problem with a fixed number of terminals can be computed in polynomial time. Notice, however, that this approach gives only the optimal time, and not the flow.

We give a polynomial time algorithm that for a fixed number of terminals finds the quickest flow. First we compute the optimum time T using the observation of Klinz [9]. Then we use the lexicographic maximum dynamic flow algorithm in Section 5 for every ordering of the sources and sinks. We claim that a feasible dynamic flow with time horizon T that sends at least v_s flow out of every source s and puts at least v_t flow into every sink t can be obtained by taking a convex combination of the $k!$ lexicographic maximum dynamic flows. To see this, consider the polytope $\mathcal{P}$ in the k-dimensional space of all terminal flow amounts: $v \in \mathcal{P}$ if $v_s \geq 0$ for every source s and $v_t \leq 0$ for every sink t and there exists a feasible dynamic flow with time horizon T that produces excess v_x at terminal x. The results of Minieka [11] and Megiddo [10] imply that the vertices of this polytope can be obtained by static lexicographic maximum flow computations on the time expanded graph $G(T)$. By also considering lexicographic maximum dynamic flow problems for all subsets of $\mathcal{D}$, we can compute a convex combination that exactly satisfies the demand at each terminal.

Theorem 6.1 For any fixed k and time T, if there exists a solution to the quickest flow problem with k terminals in time T, then the above algorithm finds one in time $O(M(m,n))$.

In a forthcoming paper, we give two extensions of the results in this section. Notice that the dynamic network flow problem considered here is equivalent to a transportation problem in the time expanded graph $G(T)$. Therefore, if the problem is integral and feasible, there must be an integral solution. Our solution, however, is fractional: the algorithm takes the integral lexicographic maximum dynamic flows and finds an appropriate convex combination. Recently we have discovered a more involved algorithm that finds an integral solution. We have also extended these results, by using the ellipsoid method, to obtain a polynomial time algorithm for the (fractional or integral) evacuation problem with a variable number of sources.

Acknowledgement

We are grateful to Bettina Klinz for valuable discussions of the quickest flow problem with a fixed number of sources.

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

[5] Fredman, M.L. and Tarjan, R.E., Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms, *Journal for the Assoc. for Comp. Mach.* 34:3(1987)596–615.

[6] Gale, D., Transient Flows in Networks, *Michigan Mathematical Journal* 6(1959)59–63.

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

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

[9] Klinz, B., Personal communication.

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

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

[12] Orlin, J.B., A Faster Strongly Polynomial Minimum Cost Flow Algorithm, *Proc. 20-th Annual Symp. Theory of Computing*, 1988, 377–387.

[13] 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).

[14] Wilkinson, W.L., An Algorithm for Universal Maximal Dynamic Flows in a Network, *Operations Research* 19:7(1971)1602–1612.

[15] Zadeh, N., A Bad Network Problem for the Simplex Method and Other Minimum Cost Flow Algorithms, *Mathematical Programming* 5(1973)255–266.