Universally maximum flow with piecewise-constant capacities *

L. Fleischer†

September 1, 1998

Abstract

A *maximum dynamic flow* generalizes the standard maximum flow problem by introducing time. The object is to send as much flow from source to sink in T time units as possible. A related problem is the *universally maximum flow*, which asks to send a flow from source to sink that maximizes the amount of flow arriving at the sink by time t for all $t \leq T$. We consider a further generalization of this problem that allows arc and node capacities to change over time. In particular, given a network with arc and node capacities that are piecewise constant functions of time with at most k breakpoints, and a time bound T, we show how to compute a flow that maximizes the amount of flow reaching the sink in all time intervals $(0, t]$ for all $0 < t \leq T$, in $O(k^2 mn \log(kn^2/m))$ time. The best previous algorithm requires $O(nk)$ maximum flow computations on a network with $(m+n)k$ arcs and nk nodes.

1 Introduction

In the 1960's, Ford and Fulkerson introduced dynamic network flows to include time in the standard network flow model. Since then, dynamic network flows have been used widely to model network-structured, decision-making problems over time: problems in electronic communication, production and distribution, economic planning, cash flow, job scheduling, and transportation. For examples, see the surveys of Aronson [1] and Powell, et al. [12].

A *maximum dynamic flow* generalizes the standard maximum flow problem by introducing time. A standard network consists of a set of nodes V and a set of arcs E which is a subset of $V \times V$. The capacity function from the arcs to the real numbers bounds the amount of flow allowed on each arc. In a *dynamic network*, the capacity function u limits the rate of flow into an arc. In addition, a dynamic network has a transit-time vector ϱ associated with the arcs. The transit time of an arc is the amount of time it takes for flow to travel from one end of the arc to the other.

*Columbia University Computational Optimization Research Center Technical Report TR-98-02.
†email: lisa@ieor.columbia.edu. Department of Industrial Engineering and Operations Research, Columbia University. Supported in part by an American Association of University Women Educational Foundation Selected Professions Fellowship and by NSF through grant DMS 9505155.

Ford and Fulkerson [3] consider the dynamic maximum flow problem: given a dynamic network with a specified source and sink, determine the maximum amount of flow that can be sent from source to sink in T time units. They show that this problem can be solved in three steps: First, compute a minimum cost circulation in the graph with transit times interpreted as per-unit-flow costs, and with an additional arc from the sink to the source with cost $-T$ and infinite capacity. Second, decompose the minimum cost flow restricted to the original graph into paths and cycles. Third, for each path in the decomposition, send flow on the path from time 0 until the time that flow sent from the source along this path will no longer reach the sink. In other words, if the length of the path is ϱ, then flow is sent along this path from time 0 until time $T - \varrho$, reaching the sink in the interval $(\varrho, T]$.

A generalization of dynamic maximum flows, a *universally maximum flow* is a flow that simultaneously maximizes the amount of flow reaching a specified sink node d, sent from a specified source node s, by time t, for all $0 < t \leq T$. Wilkinson [14] and Minieka [9] showed that such a flow exists, and they both provide algorithms to solve this problem, but these do not run in polynomial time. There is no known polynomial time algorithm that solves the universally maximum flow problem. It is not even known if the optimal solution is polynomial in the size of the input. Hoppe and Tardos [7, 8] present an $O(m\epsilon^{-1}(m + n \log n) \log U)$ algorithm that computes a dynamic flow with the property that the quantity of flow reaching the sink by time t is within $(1 - \epsilon)$ of the maximum possible for all $1 \leq t \leq T$.

When all transit times are zero and all capacity functions are constant, the universally maximum flow problem is solved by sending flow at the rate of a maximum flow in the static network at every moment of time in $[0, T]$. (The *static network* is the interpretation of the dynamic network as a standard network. Transit times are ignored, and capacities are interpreted as bound on the total flow allowed on an arc.)

Another generalization of the maximum dynamic flow problem is the problem with multiple sources and sinks The *dynamic transshipment problem* is a single commodity flow problem that asks to send specified supplies located at source nodes to satisfy specified demands at sink nodes within a given time bound T. A *universally quickest transshipment* is a dynamic flow that satisfies as much demand as possible in every interval of time $(0, t]$ for $0 < t \leq T$. If there is more than one sink, a universally quickest transshipment may not exist, so the universally quickest transshipment problem refers to a dynamic transshipment problem with one sink only. The zero transit time version of this problem is discussed in [2, 6, 10, 13]. Hajek and Ogier [6] describe the first polynomial time algorithm to find a universally quickest transshipment for a dynamic network with all zero transit times. Their algorithm uses n maximum flow computations on the underlying static network. Fleischer [2] improves upon this by describing an algorithm that solves this problem in the same asymptotic time as a preflow-push maximum flow algorithm.

In this paper, we consider the generalization of the zero transit time, universally maximum flow problem that allows both arc capacities and node storage capacities to be piecewise-constant functions on $[0, T]$ with at most k breakpoints. As with the simpler universally maximum flow problem, it is not clear that such a flow exists. Ogier [11] proves it does, and provides a polynomial time algorithm to solve this problem. Ogier proves that when the capacity functions are piecewise constant with at most k breakpoints, the maximum number of breakpoints a universally maximum flow may have is nk. These breakpoints can be computed with nk maximum flow computations on a network with nk vertices and $(m+n)k$ arcs. After the breakpoints are computed, additional maximum flow computations on the same network are used to calculate the flow between two breakpoints. Thus the total run time of Ogier's algorithm is determined by the time to solve $O(nk)$ maximum flow problems on a network with nk vertices and $(n+m)k$ arcs. As noted by Ogier [11], this problem also generalizes the universally quickest transshipment problem.

The main contributions of this paper is to a) recognize that these problems can be solved by solving a parametric maximum flow problem on a suitably defined graph and b) generalize the parametric maximum flow algorithm of Gallo, Grigoriadis, and Tarjan [4] to fit the needs of this problem. The end result is that all the computations described in [11] can be performed in the same asymptotic time as one preflow-push maximum flow computation on a network with nk vertices and $(n+m)k$ arcs. This improves the previous strongly polynomial run time by a factor of $O(nk)$.

1.1 The model

The universally maximum flow problem with piecewise constant capacities (UMFP) is defined on a network with n nodes and m arcs, along with a nonnegative, left-continuous, piecewise-constant capacity function $u : (E \cup V) \times (0, T] \to \mathcal{Z}_+ \cup \{0\}$. When referring to the capacity of a particular arc e or a particular node i at a particular time t, we use the notation $u_e(t)$ or $u_i(t)$. The vector of all capacities at time t is simply denoted $u(t)$, the vector of just arc capacities is denoted $u_E(t)$, and the vector of node capacities is denoted $u_V(t)$. If $\emptyset \neq A \subset V$, we use the notation $u_{A,\overline{A}}(t)$ to denote $\sum_{i \in A, j \in \overline{A}} u_{ij}(t)$.

For convenience of notation, we assume that breakpoints are exactly one unit apart. If this is not the case, we can scale time between each breakpoint to create an equivalent problem with this property. That is, if the time between two breakpoints is τ, then we create an equivalent problem by changing this to 1 and multiplying all capacities in this interval by τ. Any flow that solves this modified problem can be transformed back to a solution to the original problem by dividing the rate of flow sent in the new time interval by τ and sending it over an interval that is τ times longer.

In such a manner, we scale T so that it equals k. Thus, for the remainder of the paper, we assume our time bound is k, all k breakpoints are in the set $\{1, 2, \ldots, k\}$, the domain of u is $(E \cup V) \times (0, k]$, and $u(\tau) = u(\lceil \tau \rceil)$ for all $0 \leq \tau \leq k$.

Define x to be a feasible dynamic flow if x_e is a Lebesgue-measurable function on $(0, k]$ for all $e \in E$, x obeys the *arc capacity constraints* $x_e(t) \leq u_e(t)$ for all $e \in E$ and all $t \in (0, k]$, and the excess at each node, defined by

$$p_j(\tau) := \int_0^{\tau} \sum_{i \in V} x_{ij}(t)dt,$$

obeys the *node capacity constraint* $0 \leq p_j(t) \leq u_j(t)$ for all $j \in V - \{s, d\}$, and all $0 \leq t \leq k$. Denote the set of feasible dynamic flows by D. The value of flow x at time τ, denoted $v_\tau(x)$, is the total flow reaching the sink by time τ. That is,

$$v_\tau(x) := \int_0^{\tau} x_{N,d}(t)dt.$$

A *τ-maximum flow* is a feasible dynamic flow x that maximizes $v_\tau(x)$. Ogier [11] extends the traditional, static max flow-min cut theorem to τ-maximum flows with piecewise-constant capacity functions, using dynamic cuts. A *dynamic cut* is a function C that maps $[0, k]$ to subsets of vertices of the network $\mathcal{N}$ such that $C(\alpha) = C(\beta)$ for all $\alpha, \beta \in (\theta, \theta + 1]$, $\theta \in \{0, 1, \ldots, k - 1\}$, and $C(t) \cap \{s, d\} = \{d\}$ for all $t \in (0, k]$. Let $\mathcal{C} := \{C | C \text{ is a dynamic cut}\}$. Since the range of a dynamic cut is a finite set, and, over the domain $[0, k]$, each dynamic cut can change only k times (each is uniform over each interval $(\theta, \theta + 1]$), $\mathcal{C}$ is a finite set. The value of dynamic cut C at time τ is

$$w(\tau, C) = \int_0^{\tau} u_{\overline{C(t)}C(t)}(t)dt + \sum_{\theta=1}^{\lceil \tau \rceil - 1} \sum_{j \in C(\theta+1) \cap \overline{C(\theta)}} u_j(\theta). \tag{1}$$

Theorem 1.1 (Ogier [11]) $\min_{C \in \mathcal{C}} w_\tau(C) = \max_{x \in D} v_\tau(x)$.

Theorem 1.1 implies that among the finite set of dynamic cuts there is one that proves the optimality of a τ-maximum flow. Dynamic cuts can be interpreted as cuts in a modified time-expanded networks of $\mathcal{N}$. A *time-expanded network* of $\mathcal{N}$ is a directed graph that contains a copy of $\mathcal{N}$ for each time unit, and *holdover arcs* from a copy of a node at time θ to the copy of the same node at time $\theta + 1$ for $\theta \in \mathcal{Z}$. Let G_τ be the time-expanded graph on $k + 1$ copies of $\mathcal{N}$, labeled $\mathcal{N}_0, \ldots, \mathcal{N}_k$. Let $i(\theta)$ be the copy of vertex i in copy $\mathcal{N}_\theta$. Assign the arcs in $\mathcal{N}_\theta$ capacities $u_E(\theta)$ for $\theta \in \{0, \ldots, \lceil \tau \rceil\}$, and 0 otherwise. Assign holdover arcs leaving $\mathcal{N}_\theta$ capacities equal to $u_V(\theta+1)$ for $\theta = 0$ to $\lfloor \tau \rfloor$, and zero otherwise. By our scaling assumption, there are exactly $k+1$ unique graphs G_τ, one for each $\theta \in \{0, 1, \ldots, k\}$, such that for all $0 \leq \tau \leq k$, $G_\tau = G_{\lceil \tau \rceil}$. If $\theta \in \{1, 2, \ldots, k\}$, then the value of a θ-maximum flow equals the value of the minimum cut in G_θ separating all sources from all sinks.

For each real $0 \le \tau \le k$, we also define another time-expanded graph, H_τ. H_τ has the same vertex set and support as G_τ, but a slightly different capacity function. In H_τ the capacities of the arcs in $\mathcal{N}_{\lfloor\tau\rfloor+1}$ are multiplied by $\tau - \lfloor\tau\rfloor$, which is always less than one. The capacities of all other arcs equal the capacities of the arcs in G_τ. Denote the capacity function of H_τ by $\tilde{w}_\tau$. Note that $H_\theta = G_\theta$ for all $\theta \in \{0, 1, \dots, k\}$.

The value of a dynamic cut at time τ corresponds to the value of the arcs entering the sink side of a cut containing all copies of the sink in H_τ. We make this explicit as follows. Denote by V^*_τ the vertex set of H_τ (and G_τ). Since the vertex sets of each H_τ and G_τ are the same for all τ, we will denote this V^* when the context is clear. A *cut* in H_τ is a set $R \subset V^*$ such that R contains all copies of the sink, and no copies of the source. That is, if s_θ is the copy of s and d_θ is the copy of d in N_θ for $\theta \in \{0, 1, \dots, k\}$, then $R \cap \bigcup_{\theta=0}^{k} \{s_\theta, d_\theta\} = \bigcup_{\theta=0}^{k} \{d_\theta\}$.

For a set $R \subset V^*$, define $\tilde{w}_\tau(R)$ to be the sum of capacities of arcs in H_τ entering R. We can equate each dynamic cut $C \in \mathcal{C}$ with a cut R such that $w_\tau(C) = \tilde{w}_\tau(R)$ by setting $R \cap \mathcal{N}_\theta = C(\theta)$ for $\theta \in \{0, 1, \dots, k\}$. Define $R(\theta) := R \cap \mathcal{N}_\theta$ for $\theta \in \{0, 1, \dots, k\}$.

Corollary 1.2 $\max_{x\in D} v_\tau(x) = min_{R\subset V^*} \tilde{w}_\tau(R)$. ■

By the submodularity of cut functions in a graph, the non-empty intersection of minimum cuts is a minimum cut. Thus, there is a minimum cut in H_τ with smallest sink-side. Let R_τ be the minimum cut such that $R_\tau \subseteq R$ for all R such that $\tilde{w}_\tau(R) = \tilde{w}_\tau(R_\tau)$. The following lemma was proved first by Ogier [11]. We include a slightly different proof to provide some understanding of the choice of definitions for R_τ.

Lemma 1.3 *For all $0 \le \tau < \sigma \le k$, $R_\tau \subseteq R_\sigma$, and R_τ is a left continuous function of τ.*

Proof: We show that if $\tau < \sigma \le \lfloor\tau\rfloor + 1$ then $R_\tau \subseteq R_\sigma$, which, using an inductive argument, implies the first part of the lemma. Suppose that for all ν such that $\tau \le \nu \le \sigma$, $R_\nu = R_\tau$ or $R_\nu = R_\sigma$. Consider R_τ and R_σ as cuts in $H_{\lfloor\tau\rfloor+\lambda}$ for $0 \le \lambda \le 1$. Let $\lambda_1 = \tau - \lfloor\tau\rfloor$ and $\lambda_2 = \sigma - \lfloor\sigma\rfloor = \sigma - \lfloor\tau\rfloor$. For simplicity of notation, let $\mathbf{w}_\lambda = \tilde{w}_{\lfloor\tau\rfloor+\lambda}$. The capacity of cuts R_τ and R_σ under $\mathbf{w}_\lambda$ increase linearly in λ for $0 \le \lambda \le 1$. By definition of R_τ and R_σ we have that $\mathbf{w}_{\lambda_1}(R_\tau) \le \mathbf{w}_{\lambda_1}(R_\sigma)$, and $\mathbf{w}_{\lambda_2}(R_\tau) \ge \mathbf{w}_{\lambda_2}(R_\sigma)$. Thus there is a λ_{12}, $\lambda_1 \le \lambda_{12} \le \lambda_2$, such that $\mathbf{w}_{\lambda_{12}}(R_\tau) = \mathbf{w}_{\lambda_{12}}(R_\sigma)$. By submodularity of cut functions in a graph, this implies that $\mathbf{w}_{\lambda_{12}}(R_\tau) = \mathbf{w}_{\lambda_{12}}(R_\tau \cap R_\sigma)$. Since R_τ is the smallest minimum cut for τ, this implies that $R_\tau = R_\tau \cap R_\sigma$, i.e. $R_\tau \subseteq R_\sigma$. This in turn implies that $R_\tau = R_{\lfloor\tau\rfloor+\lambda_{12}}$, and hence that R_τ is a left continuous function of τ. ■

Let W be the set of breakpoints of R_τ as a function of τ. That is, W is the set of values τ such that $R_\tau \ne R_{\tau+}$. Ogier's main contribution was to prove that there is an optimal solution to UMFP that is uniform on the intervals between successive breakpoints of W, and that such an optimal flow can be defined piecemeal by special τ-maximum flows.

Let x^τ be a τ-maximum flow that is constant on each interval $(\theta-1,\theta]$ for $\theta \in \{1,2,\dots,\lfloor\tau\rfloor\}$ and on interval $(\lfloor\tau\rfloor,\tau]$, with excess function p^τ that satisfies $p^\tau_j(\tau)=0$ for all $j \in V-\{s,d\}$. That such a flow exists is implied by Corollary 1.2: Compute a maximum flow in H_τ. This flow saturates the arcs entering R_τ, hence has the value of a τ-maximum flow. It does not use any node storage arc leaving $\mathcal{N}(\lceil\tau\rceil)$, hence in the dynamic setting, completes by time τ. Define x^0 to be the dynamic flow such that for all t, $f=x^0(t)$ is a maximum flow for static network $\mathcal{N}$ with capacities $u(t)$, sources $\overline{R_t(\lceil t\rceil)}$, and sink d. Theorem 1.4 below extends the static flow concept of complementary slackness to this problem.

For each node j of the dynamic network $\mathcal{N}$, define

$$\begin{aligned} Q_j(t) &:= \{\tau \mid \tau \geq t, j \in \overline{R_\tau(\lceil t\rceil)}\} = \{\tau \mid \tau \geq t, j(\lceil t\rceil) \in \overline{R_\tau}\}. \\ q_j(t) &:= \begin{cases} \max\{\tau \mid \tau \in Q_j(t)\}, & Q_j(t) \neq \emptyset \\ 0, & Q_j(t) = \emptyset \end{cases} \end{aligned} \tag{2}$$

Since R_τ is left-continuous, $q_j(t)$ is well-defined.

Theorem 1.4 (Ogier [11]) *Flow x^*, defined by*

$$x^*_{ij}(t) = \begin{cases} x^\tau_{ij}(t), & \text{if } q_i(t) = q_j(t) = \tau \\ u_{ij}(t), & \text{if } q_i(t) > q_j(t) \\ -u_{ij}(t), & \text{if } q_i(t) < q_j(t) \end{cases}$$

is a τ-maximum flow for all $\tau \in (0,T]$.

The proof of this theorem is technical. The intuition is that if, for $t \in (\theta-1,\theta]$, there is a σ-maximum flow with $\sigma \geq t$ such that node $i(\theta)$ is on the source side of the cut defined by this flow, and $j(\theta)$ is on the sink side, then the flow on the arc (i,j) should be at capacity in this interval. If $i(\theta)$ and $j(\theta)$ are always on the same side of the cut corresponding to any τ-maximum flow, then the flow on the arc should be determined by the τ-maximum flow with the latest completion time if this time is after t, or the maximum flow on $\mathcal{N}_\theta$ otherwise.

If W is known, then x^0 can be computed with $|W|$ maximum flows in a network the same size as $\mathcal{N}$, and x^* can be computed with an additional $|W|$ maximum flows in each H_τ, for all $\tau \in W$. Ogier describes a method to compute W in a piecewise fashion, computing $W \cap (\theta,\theta+1]$ for $\theta = 0,\dots,k-1$. Define the minimum cut function $\overline{\kappa}$ by setting $\overline{\kappa}(\tau)$ equal to the value of the minimum cut in H_τ.

Lemma 1.5 *Within the domain* $(\theta, \theta+1]$ *for* $\theta \in \{0, \dots, k-1\}$, $\overline{\kappa}$ *is concave and piecewise linear with at most* $n-2$ *breakpoints.*

Proof: For $\tau \in (\theta, \theta+1]$, $\theta \in \{0, \dots, k-1\}$, $\overline{\kappa}(\tau)$ is the minimum function of a set of linear functions with positive slope: all capacities are linear, increasing functions of τ on the unit interval; hence, the value of all cuts are also linear, increasing functions of τ. This implies that $\overline{\kappa}$ is concave. Since R_τ is increasing in τ, changes in the slope of the cut function imply an increase in $|R_\tau \cap \mathcal{N}_{\theta+1}|$. This together with the fact that $d_{\theta+1} \in R_\tau$ and $s_{\theta+1} \notin R_\tau$ implies that $\overline{\kappa}(\tau)$ has at most $n-2$ breakpoints in a unit interval. ■

Ogier [11] computes each subset $W \cap (\theta, \theta+1]$ by finding τ such that $R_{\theta+}(\tau) = R_{\theta+1}(\tau)$, computing R_τ, and recursing on the subintervals $(\theta, \tau]$ and $(\tau, \theta+1]$ until the interval does not properly contain elements of W. Since $\overline{\kappa}$ is concave in the interval $(\theta, \theta+1]$, the τ that satisfies $R_{\theta+}(\tau) = R_{\theta+1}(\tau)$ lies in the interval. Lemma 1.5 implies that $|W|$ is at most nk. Ogier uses one maximum flow computation on a graph with at most nk vertices at each recursive step, yielding an $O(n^4k^4)$ algorithm to compute W. Substituting the fastest strongly polynomial maximum flow algorithm — the preflow-push algorithm of Goldberg and Tarjan [5], Ogier's approach leads to an $O(k^2 mn \log(kn^2/m))$ algorithm to compute x^*.

2 A Faster Algorithm

In this section, we discuss the main contribution of this paper which is a generalization of the parametric maximum flow algorithm of Gallo, Grigoriadis, and Tarjan [4] that speeds up the computation of W and the x^τ, $\tau \in W$. We first review the parametric maximum flow algorithm, and the preflow-push maximum flow algorithm on which it is based.

2.1 Parametric maximum flows

Gallo, Grigoriadis, and Tarjan [4] present several algorithms based on a *parametric preflow algorithm.* Given a graph G on n vertices and m arcs such that capacities of arcs leaving the source are nondecreasing functions of θ, capacities of arcs entering the sink are nonincreasing functions of θ, and all other capacities are constant, the parametric preflow algorithm finds a maximum flow and a minimum cut for each value of θ in an increasing sequence $\theta_1 < \ldots < \theta_k$. The sequence may be computed on-line.

The parametric preflow algorithm is a parameterized version of the Goldberg and Tarjan [5] preflow-push algorithm for computing maximum flows. The preflow-push algorithm maintains at all times a feasible preflow and a valid labeling. A *feasible preflow* is a flow f satisfying arc capacity constraints $f_{ij} \leq u_{ij}$ and relaxed node conservation constraints $\sum_{i \in V} f_{ij} \geq 0$ for all $j \in V - \{s\}$. The quantity $\sum_{i \in V} f_{ij}$, when strictly positive, is called the *excess* at node j. A *valid labeling* l for preflow f is a function from the vertices to the nonnegative integers satisfying $l(s) = n$, $l(d) = 0$, $l(i) \geq 0$, and $l(i) \leq l(j) + 1$ for all (i, j) such that $f_{ij} < u_{ij}$.

The preflow-push algorithm typically starts with the preflow defined by saturating all arcs leaving the source, and the labeling $l(i) = 0$ for all $i \neq s$. Throughout the course of the algorithm, labels of nodes may increase but never decrease. The algorithm terminates when no nodes besides the source or sink contain excess. The entire analysis of the algorithm depends on the fact that the labels can only increase, and that at all times there is a path of residual arcs from any vertex to either the source or the sink, implying that no node has label greater than $2n$. The bound on the run time of the algorithm is determined by bounding the number of times each node is relabeled.

The analysis of this algorithm remains valid as long as it starts with any feasible preflow and corresponding valid labeling. There is nothing that requires the algorithm to start with the stated preflow and labeling. This fact is exploited by Gallo, et al. in their generalization of this algorithm to solve the parametric maximum flow problem.

The parametric preflow algorithm starts with capacities determined by the smallest value of the parameter θ, i.e. θ_1, and computes a maximum flow. Arc capacities are then increased to the next largest value of θ. If there are arcs from s to nodes with label less than $l(s)$, the flow on these arcs is increased to meet the new capacities. The flow on arcs entering the sink is decreased, if necessary, to meet the new capacities, increasing the excess at the nodes adjacent to the sink. This results in a new, feasible preflow for which the previous labeling is still valid, since all saturated arcs in the previous preflow remain saturated in the new preflow. Thus the preflow-push algorithm will compute a maximum flow in this network with updated capacities.

The same bound on the run time of the preflow-push algorithm is also a bound on the run time of the parametric preflow push algorithm: $O(nm \log(n^2/m))$ time [4, 5] (assuming the number of values of θ is not more than $m \log(n^2/m)$). This is because the bound on the run-time of the preflow-push algorithm depends only on the number of times a node is relabeled, node labels never decrease in either algorithm, and all node labels are $\leq 2n$.

Using this parametric preflow algorithm, Gallo, Grigoriadis, and Tarjan [4] describe an algorithm that finds all breakpoints of $\kappa(\theta)$, the minimum cut function of graph G, in the same asymptotic time. Like Ogier's algorithm, this algorithm relies on the concavity of κ. Once the breakpoints are found, the parametric preflow algorithm can be invoked again to compute the maximum flows and minimum cuts corresponding to these breakpoints.

2.2 Generalizing the parametric maximum flow algorithm to solve UMFP

We generalize the parametric preflow algorithm and the breakpoint algorithm to solve the universally maximum flow problem with piecewise constant capacities. This enables us to reduce the time needed to compute W and the x^τ, $\tau \in W$ defined in Section 1.1 to $O(k^2nm\log(kn^2/m))$. Since x^0 can be computed in $O(knm\log(n^2/m))$ time using k iterations of Goldberg and Tarjan's push-relabel maximum flow algorithm [5], this implies that x^* can also be computed in $O(k^2nm\log(kn^2/m))$ time.

Theorem 2.1 *A universally maximum flow with piecewise-constant capacities can be constructed in $O(k^2mn\log(kn^2/m))$ time.*

Proof: Our algorithm integrates the work of Gallo, et al. [4] into the framework of the Ogier algorithm. In Step 1, we use the parametric preflow algorithm of Gallo, et al. to compute the minimum cuts R_θ and θ-maximum flows x^θ in H_θ for $\theta = 0, 1, \ldots, k$. In Step 2, we generalize the breakpoint algorithm of Gallo, et al. to compute the minimum cuts R_τ and corresponding maximum flows for $\tau \in (\theta, \theta+1] \cap W$, for each $\theta = 0, \ldots, k-1$.

We consider a parametric flow problem based on the graphs H_τ. Instead of considering the graphs H_τ for $\tau \in (0, k]$ as separate graphs, we consider one graph H on the same vertex set but with parameterized capacities, so that the capacities of arcs of H at time τ equal the capacities of H_τ. That is, $H(\tau) = H_\tau$. More precisely, using $\mathcal{N}_\theta$ to denote the θ^{th} copy of network $\mathcal{N}$ in H, an arc in $\mathcal{N}_\theta$ has capacity 0 for $0 \le t \le \theta - 1$, capacity $(t - \theta + 1)u_e(\theta)$ for $\theta - 1 < t \le \theta$, and capacity $u_e(\theta)$ for $\theta < t \le k$. An arc from $\mathcal{N}_\theta$ to $\mathcal{N}_{\theta+1}$ has capacity 0 for $0 \le t \le \theta$ and capacity $u_j(\theta)$ for $\theta < t \le k$.

Step 1: Computing R_θ and x^θ for $\theta \in \{0, 1, \ldots, k\}$. To compute the θ-maximum flows x^θ, $\theta \in \{0, \ldots, k\}$, we construct a parametric flow problem on a modified G_k $(= H_k)$. Recall that s_i is the copy of the source in $\mathcal{N}_i$, and d_i is the corresponding sink. We introduce a super source s^S with infinite capacity arcs (s^S, s_i) for each $i \in \{0, \ldots, k\}$, a super sink d^S with arcs (d_i, d^S) for each $i \in \{0, \ldots, k\}$ with capacity function that is infinite when $i \le \theta$ and zero otherwise, and call this parametric network $\hat{G}_k$. We then solve the parametric flow problem in $\hat{G}_k$ to find the maximum flows corresponding to each parameter $\theta \in \{0, 1, \ldots, k\}$. These maximum flows are the flows x^θ for $\theta \in \{0, 1, \ldots, k\}$. This is because they correspond to cuts in $H(\theta) \equiv H_\theta$ of the same value; and, as a cut in H_θ, they keep this value even when all sinks d_i that are not in the original sink side of the cut are moved to the sink side: If d_i is not in the sink side of the cut for parameter θ in $\hat{G}_k$, the capacity of (d_i, d^S) for parameter θ must be finite, and is therefore zero. Thus the capacity of all arcs entering d_i in H_θ is also zero, and d_i can be moved to the sink side of the cut.

To solve this parametric flow problem, we reverse $\hat{G}_k$ by reversing the direction of every arc in $\hat{G}_k$, and apply the algorithm in [4] to this network. This parametric flow problem is of the form that solved in [4]: all arcs that vary with θ are leaving the source, and the capacities are all increasing functions of θ. Thus, x^θ and R_θ for all $\theta \in \{0, \ldots, k\}$ can be computed in the same asymptotic time as one maximum flow computation in the graph G_k: $O(k^2 nm \log(kn^2/m))$ time.

Step 2: Computing R_τ and x^τ for all $\tau \in W$. To find the elements of W, and the corresponding τ-maximum flows within interval $(\theta, \theta+1]$, we generalize the version of parametric maximum flow algorithm [4] that finds all breakpoints of the minimum cut function $\overline{\kappa}(\theta)$. We start with R_θ and $R_{\theta+1}$. The corresponding graphs are $H(\theta)$ and $H(\theta+1)$. As τ increases from θ to $\theta+1$, the capacities of the arcs in $H \cap \mathcal{N}_{\theta+1}$ increase linearly from 0 to $u_E(\theta+1)$. Because the change in the capacity of the arcs is linear, the change in the minimum cut function is piecewise linear. For the correctness and speed of their algorithm, Gallo et al. require that all parameterized arcs either leave the source or enter the sink, and this does not hold here. However, the Gallo et al. requirements as stated above are merely sufficient conditions. The following conditions are also sufficient, but more general [4].

1. After increasing θ, it must be possible to adjust the the flow so that it is a feasible preflow and modify the labeling so that it remains valid without decreasing any labels.

2. The minimum cut function $\kappa(\tau)$ must be concave.

The second item is necessary for us to find the elements of W efficiently using a modified version of the Goldberg and Tarjan maximum flow algorithm and holds true here by Lemma 1.5. The first item is necessary for us to compute all corresponding maximum flows and minimum cuts in the same asymptotic time as one — this is a requirement of the parametric preflow algorithm as discussed above. In the next paragraph, we explain how to satisfy this first item.

We run an iteration of the breakpoint algorithm in the reverse network H^R of H, restricted to the time interval $(\theta - 1, \theta]$ for some $\theta \in \{1, 2, \ldots, k\}$. After computing the first maximum flow, we increase the current value of θ by ϵ, thus the increasing the capacity of arcs in $\mathcal{N}_\theta$ by ϵ. (Here, ϵ is determined by the breakpoint algorithm in [4].) To adjust the preflow, we then increase the current flow in $\mathcal{N}_\theta$ by $\epsilon \frac{x_{ij}(\theta)}{u_{ij}(\theta)}$ for each arc $(i(\theta), j(\theta))$. Recall that $x_{ij}(\theta)$ equals the flow on arc (i,j) in $\mathcal{N}_\theta$. Since there are no arcs with positive capacity that enter $\mathcal{N}_\theta$ in H^R, there is no node in $\mathcal{N}_\theta$ that receives flow from outside $\mathcal{N}_\theta$. Hence this adjustment of the preflow maintains non-negative excess within $\mathcal{N}_\theta$, and the adjusted preflow is feasible. Since this update keeps previously saturated arcs saturated and empty arcs empty, the current labeling is still valid.

The breakpoint algorithm is repeated for each unit interval in $[0, k]$. This could take k parametric maximum flow computations in H^R, i.e. k times $O(k^2 nm \log(kn^2/m))$ time. To speed this up, we use the fact that we have already computed R_θ for all $\theta \in \{1, \ldots, k\}$ in Step 1, and that, by Lemma 1.3, $R_\theta \subseteq R_\tau \subseteq R_{\theta+1}$, for all $\theta \leq \tau \leq \theta + 1$. Thus, in searching for breakpoints of $\kappa(\tau)$ in the interval $(\theta, \theta + 1]$, we can restrict the computations performed for each unit interval to the subgraph of H^R on the vertex set $R_{\theta+1} \backslash R_\theta \cup \{v_1, v_2\}$, where v_1 is the vertex resulting from the contraction of all vertices in $\overline{R_{\theta+1}}$ and v_2 is the vertex resulting from the contraction of R_θ.

We also need to compute the τ-maximum flows x^τ for all $\tau \in W$. We don't actually need to know the flow on every arc at every time step. By Theorem 1.4, it is sufficient to determine the values $x^\tau_{ij}(t)$ for t and τ such that $q_i(t) = q_j(t) = \tau$. Since x^τ is constant on unit intervals, (2) implies that it is necessary to compute $x^\tau_{ij}(t)$ only for $t \leq \tau$ where $i(\lceil t \rceil), j(\lceil t \rceil) \in \overline{R}_\tau$ and $i(\lceil t \rceil), j(\lceil t \rceil) \in R_\sigma = R_{\tau+}$. All of these arcs are contained in the subgraph of H^R induced by the vertices in $R_\sigma \backslash R_\tau$. In order to compute x^τ restricted to this space-network interval, we must also specify the flow on all arcs entering and leaving this interval, i.e. all arcs adjacent to v_1 and v_2. The flow on such arcs that are adjacent to R_τ in the relevant time interval is determined by definition of R_τ: all arcs entering R_τ must be saturated in x^τ, and all arcs leaving R_τ must be empty in x_τ. The flow on the remainder of these arcs, i.e. those adjacent to $\overline{R_\sigma}$, is similarly determined by citing one additional lemma of Ogier's.

Lemma 2.2 (Ogier [11]) *For each $\tau \in (0, T)$, $R_{\tau+}$ is a τ-minimum cut.*

In particular, for consecutive breakpoints $\tau < \sigma$ in W, $R_{\tau+} = R_\sigma$ so R_σ is also a τ-minimum cut. Thus the above observations for arcs entering and leaving R_τ also hold for R_σ, and now we can compute x^τ on H^R restricted to $R_\sigma \backslash R_\tau$.

Performing maximum flow computations on k graphs that contain a sum total of nk vertices and $mk + nk$ arcs is bounded by the time to compute a maximum flow on a single graph with nk vertices and $(m + n)k$ arcs. Thus the total run time is $O(k^2 nm \log(kn^2/m))$. ∎

References

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

[2] L. Fleischer. Faster algorithms for the quickest transshipment problem with zero transit times. In *Proceedings of the Ninth Annual ACM/SIAM Symposium on Discrete Algorithms*, pages 147–156, 1998. Submitted to SIAM Journal on Optimization.

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

[4] 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.

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

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

[7] B. Hoppe. *Efficient Dynamic Network Flow Algorithms.* PhD thesis, Cornell University, June 1995. Department of Computer Science Technical Report TR95-1524.

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

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

[10] F. H. Moss and A. Segall. An optimal control approach to dynamic routing in networks. *IEEE Transactions on Automatic Control*, 27(2):329–339, 1982.

[11] R. G. Ogier. Minimum-delay routing in continuous-time dynamic networks with piecewise-constant capacities. *Networks*, 18:303–318, 1988.

[12] 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.

[13] G. I. Stassinopoulos and P. Konstantopoulos. Optimal congestion control in single destination networks. *IEEE transactions on communications*, 33(8):792–800, 1985.

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