Lexicographic Flow

Dexter Kozen
Department of Computer Science
Cornell University
Ithaca, New York 14853-7501, USA

June 25, 2009

Abstract

The lexicographic flow problem is a flow problem in which the edges are assigned priorities, and we wish to find a flow that is lexicographically maximum with respect to the priority assignment. The problem is reducible to a weighted flow problem, but we show that exponentially large weights are necessary in general. We then give an efficient direct algorithm that does not use weights.

1 Introduction

There is a well-studied class of flow problems known variously as the *transportation problem*, *transshipment problem*, or *circulation problem* [1, 2, 3, 4, 5]. A problem instance is specified by a pair of disjoint finite sets U, V called *producers* and *consumers* (or *sources* and *sinks*), respectively; an edge relation $E \subseteq U \times V$; and a *capacity function* bounding the amount of flow that can be produced by each source and consumed by each sink.

In this paper we are interested in a variant of this problem in which the edges are ordered by priority, and we wish to find a flow that is lexicographically maximum with respect to priority. That is, as much flow as possible should be assigned to the edges of highest priority; then fixing this value, as much flow as possible should be assigned to the edges of the next highest priority; and so on. This is called the *lexicographic flow problem*. A lexicographically max flow is not necessarily a max flow, nor vice-versa.

The problem is not to be confused with the unrelated problem of a similar name studied in [6]. The lexicographic flow problem can be reduced to a weighted flow problem, but it is not immediately clear how to choose the weights. We show in Section 3 that exponentially large weights are necessary. It is also easy to show that a straightforward iterative greedy algorithm—find a max flow on the edges

of highest priority, form the residual graph, find a max flow on the edges of next highest priority, form the residual graph, and so on—does not work.

The problem arises in modeling the migration of North American migratory birds. In this application, we are given individual static observations of a bird species at various locations throughout the year, and we wish to infer dynamic migration patterns. Here U and V are copies of the same set, representing physical locations in North America. The capacity of a source point $u \in U$ is the population at u on one day and the capacity of a sink point $v \in V$ is the population at v on the next day. The edges form a complete bipartite graph $E = U \times V$. The partition elements are the equivalence classes of equidistant pairs, with closer pairs of higher priority; thus a $\leq_{\text{lex}}$-maximum max flow is one that favors shorter distances. The solution of the lexicographic flow problem gives a reasonable model of the migration paths from one day to the next.

In Section 2, we formulate the lexicographic flow problem. In Section 3, we show how to choose edge weights so as to reduce the problem to a weighted flow problem. We determine necessary and sufficient conditions on the weights for a maximum-weight flow to give a lexicographically maximum flow independent of the choice of edge set, capacity function, and priority assignment. We show that exponentially large weights are necessary in general (which is prohibitive for our application to bird migration). Finally, in Section 4, we give an efficient iterative flow-based algorithm that does not use weights.

2 Problem Statement

A problem instance is specified by a tuple (U, V, E, c, M), where (U, V, E) is a bipartite graph, $E \subseteq U \times V$, $c : U \cup V \to \{\alpha \in \mathbb{R} \mid \alpha > 0\}$ is the *capacity function*, and $M \subseteq U \cup V$ is the set of *anchors*.

The sets U, V are called *producers* and *consumers* (or *sources* and *sinks*), respectively. The capacity function c bounds the amount of flow that can be produced by each source $u \in U$ and consumed by each sink $v \in V$.

The anchors M are included for technical reasons that will become clear. They are used to specify lower bounds of a restricted form. If a node is an anchor, then a flow is not considered feasible unless it saturates that node; in other words, the flow produced at a source or consumed at a sink must equal the capacity of the node.

A *flow* is a map $f : E \to \{\alpha \in \mathbb{R} \mid \alpha \geq 0\}$. A flow is *feasible* if the capacity

constraints are satisfied; thus

$$\sum_{\substack{v\in V\\(u,v)\in E}} f(u,v) \le c(u),\ u \in U \qquad \sum_{\substack{u\in U\\(u,v)\in E}} f(u,v) \le c(v),\ v \in V$$

$$\sum_{\substack{v\in V\\(u,v)\in E}} f(u,v) = c(u),\ u \in U \cap M \qquad \sum_{\substack{u\in U\\(u,v)\in E}} f(u,v) = c(v),\ v \in V \cap M.$$

The *value* of a flow f, denoted $|f|$, is the sum of the flow value on all edges, which is the same as the amount of flow produced by all sources and also the amount of flow consumed by all sinks. A flow f is a *max flow* if $|f|$ is maximum among all flows. If there exists a flow, then there exists a max flow, which can be found efficiently by any one of several standard algorithms; see [1, 3, 5]. Moreover, if the capacities are integers, then there is an integral max flow.

The set of feasible flows forms a closed, bounded, convex polyhedron in $\mathbb{R}^E$. It is *convex* in the sense that any convex combination of flows (that is, $\sum_i \alpha_i f_i$, where the f_i are flows and the α_i are nonnegative reals such that $\sum_i \alpha_i = 1$) is again a flow. A convex combination of flows is called *non-extremal* if all coefficients α_i are nonzero.

A node is *saturated* under a flow f if the amount of flow into or out of that node is equal to the capacity of the node. Clearly, if f is a max flow, then for all $(u, v) \in E$, either u or v is saturated under f, otherwise one could increase the flow along the edge (u, v).

2.1 Lexicographic Flows

We are interested in finding a *lexicographically maximum* flow with respect to a priority assignment $\pi : E \to \mathbb{N}$, the higher numbers of higher priority. We seek a flow that prefers higher priority edges over lower priority ones.

Formally, any flow f can be decomposed as a sum

$$f = \sum_i f \restriction i, \qquad (f \restriction i)(e) \stackrel{\text{def}}{=} \begin{cases} f(e), & \text{if } \pi(e) = i, \\ 0, & \text{otherwise.} \end{cases}$$

The summands $f \restriction i$ are not necessarily feasible flows, since lower bounds may be violated.

Define $f \le_{\text{lex}} g$ if either

(i) $|f \restriction i| = |g \restriction i|$ for all i; or

(ii) $|f \restriction i| \ne |g \restriction i|$ for some i, and $|f \restriction i| < |g \restriction i|$ for the greatest such i.

We write $f \equiv_{\text{lex}} g$ in case (i) and $f <_{\text{lex}} g$ in case (ii). The relation $\leq_{\text{lex}}$ is a total preorder on the set of flows and is called *lexicographic order*. We are interested in finding a $\leq_{\text{lex}}$-maximum flow among all feasible flows.

A $\leq_{\text{lex}}$-maximum flow is not necessarily a max flow, nor vice-versa. However, if $E = U \times V$ (which is the case in our application of interest), any $\leq_{\text{lex}}$-maximum flow is automatically a max flow. Under any $\leq_{\text{lex}}$-maximum flow, either all the nodes of U are saturated or all the nodes of V are saturated; if not, there would be an edge along which the flow could be increased, giving a $\leq_{\text{lex}}$-greater flow. But if either U or V is totally saturated, then the flow is a max flow.

2.2 Linear Programming Formulation

We can formulate the problem as a linear programming problem and give an iterated solution. In fact, this might be a reasonable practical approach, since there exist very good linear programming solvers based on the network simplex method.

Let there be a real variable x_e for each edge $e \in E$. Let A_i be the set of edges of ith highest priority.

At stage k, suppose we have determined the lexicographically maximum flow values m_i on A_i for $1 \leq i \leq k-1$. To obtain m_k, maximize $\sum_{e \in A_k} x_e$ subject to the capacity constraints plus the additional constraints $m_i = \sum_{e \in A_i} x_e$ for $1 \leq i \leq k-1$.

3 Weighted Flow

It is possible to reduce the lexicographic flow problem to a weighted flow problem, but it is not immediately obvious how to choose the weights. In this section we determine necessary and sufficient conditions on the weights for a maximum-weight flow to give a lexicographically maximum flow independent of the choice of edge set, capacity function, anchor set, and priority assignment. Exponentially large weights are necessary in general.

Given a sequence of weights $w = w_1, w_2, \ldots$ and a flow f, define $w(f) \stackrel{\text{def}}{=} \sum_i w_i \, |f \restriction i|$. A *$w$-max flow* is a flow that maximizes $w(f)$.

For this analysis, we extend flows to $f : (U \times V) \cup (V \times U) \to \mathbb{R}$ by skew symmetry: $f(v,u) \stackrel{\text{def}}{=} -f(u,v)$. A flow f is *lexicographically positive*, *lexicographically nonnegative*, or *lexicographically negative* according as $f >_{\text{lex}} 0$, $f \geq_{\text{lex}} 0$, or $f <_{\text{lex}} 0$, respectively.

Lemma 3.1 *If f is lexicographically maximum and g is not, then $f - g$ can be decomposed as a sum of at most $|E|$ lexicographically nonnegative paths or cycles. At least one of these must be lexicographically positive.*

Proof. Consider the flow $f - g$. Starting from any node u, trace a path along edges of positive flow (that is, along edges e such that $f(e) > g(e)$) until encountering a node previously seen or a node with no exiting edges, and trace a path from u backwards until encountering a node previously seen or a node with no entering edges. Form a cycle or path flow of nonzero value, a cycle flow if possible, a path flow from a node with no entering edges to a node with no exiting edges if not. The value of the flow is the smallest flow value on any edge on the path or cycle. Subtract off this flow and delete edges whose labels are now 0; at least one edge vanishes. Repeat the process until there are no more edges.

Let C be the set of path and cycle flows obtained by this construction. For $D \subseteq C$, define $h_D = \sum_{h \in D} h$. Then $f = g + h_C$. For any $D \subseteq C$, $g + h_D$ and $f - h_D$ are feasible flows, because their flow values on each edge lie between $g(e)$ and $f(e)$, and both g and f are feasible flows. Moreover, every path or cycle flow $h \in C$ is lexicographically nonnegative, otherwise $f - h$ would be lexicographically greater than f, contradicting the choice of f as a lexicographically maximum flow; and at least one must be lexicographically positive, since $f = g + h_C$. □

Theorem 3.2 *Let $n = \min |U|, |V| \geq 2$. Call a weighting w* adequate *if for all edge relations E, all priority assignments $\pi : E \to \mathbb{N}$, all capacity functions c, and all anchor sets M, the w-max flows and the lexicographically maximal flows coincide. The following conditions are necessary and sufficient for adequacy:*

(i) $w_0 > 0$*; and*

(ii) $w_{i+1} - 2w_0 > n(w_i - w_0)$ *for all* $i \geq 0$.

Proof. It follows from conditions (i) and (ii) that $w_{i+1} > 2w_i > 0$ for all i, so the weights must grow exponentially.

First we show that conditions are sufficient for adequacy. Certainly $f \equiv_{\text{lex}} g$ implies $w(f) = w(g)$, so all lexicographically maximal flows have the same weight. Thus it suffices to show that if f is lexicographically maximum and $g <_{\text{lex}} f$, then $w(g) < w(f)$.

If there is only one priority class, we only need the one relevant weight to be positive, which is ensured by condition (i). So assume that there are at least two priority classes.

By Lemma 3.1, $f - g$ decomposes into a sum of lexicographically nonnegative path and cycle flows. It thus suffices to show that each lexicographically positive path or cycle flow h has positive weight. Suppose the value of h is $\alpha > 0$. To be lexicographically positive, if i is the highest priority such that $h \restriction i \neq 0$, the number of edges of priority i on the path or cycle h in $U \times V$ must exceed the number of edges of priority i in $V \times U$. For edges of priority other than i, the weight of h is minimized when edges from U to V are all of weight w_0 and edges from V to U are all of weight w_{i-1}. The length of h is at most $2n$, and there is at most one more edge in $V \times U$ than in $U \times V$. This gives a lower bound of

$w_i\alpha - nw_{i-1}\alpha + (n-2)w_0\alpha$ on the weight. Condition (ii) implies that this is positive.

Now we show that the conditions are necessary for adequacy. For condition (i), if $w_0 \leq 0$, consider the flow problem with two nodes and one edge e with $\pi(e) = 0$. Then the lexicographically max flow is of minimum weight.

Suppose now that (i) holds but (ii) is violated. Then

$$w_{i+1} - 2w_0 \;\leq\; n(w_i - w_0) \tag{3.1}$$

for some i. Consider the flow problem illustrated in Fig. 1 for $n = 4$. All

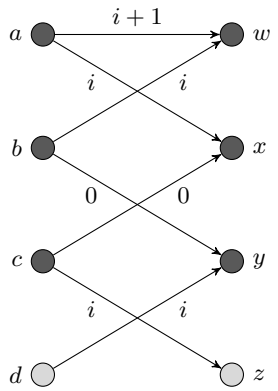

Figure 1: *A counterexample for condition* (ii)

capacities are 1 and all nodes are anchored except d and z. The lexicographically maximum flow assigns 1 to the edges (a, w), (b, y), and (c, x) and 0 to the other edges, giving weight $w_{i+1}+2w_0$, whereas the lexicographically smaller flow using edges (a, x), (b, w), (c, z), and (d, y) has weight $4w_i$, which by (3.1) is at least at great as $w_{i+1} + 2w_0$. In general, for even n, a similar figure with $2n$ nodes would have a lexicographically maximum flow of weight $w_{i+1} + (n-2)w_0$ and a lexicographically smaller flow of weight nw_i, which by (3.1) is at least at great as $w_{i+1} + (n-2)w_0$. □

4 An Iterative Flow-Based Algorithm

4.1 Overview of the Algorithm

The algorithm is iterative. Before stage i, we have anchored some nodes M_{i-1} and deleted some edges to get $E_{i-1} \subseteq E$. These modifications are such that any flow in $(U, V, E_{i-1}, c, M_{i-1})$ is forced to be lexicographically maximum on

$A_1, \ldots, A_{i-1}$, and all lexicographically maximum flows on E saturate the anchored nodes M_{i-1} and do not use edges in $E - E_{i-1}$. We find a flow that maximizes the flow on A_i, then anchor some more nodes and delete some more edges that extend the invariant to $A_1, \ldots, A_i$.

Most of the following few subsections develop the properties that will be used in each stage of this algorithm.

4.2 Critical Nodes and Useless Edges

Let (U, V, E, c, M) be a flow problem. Call $u \in U \cup V$ *critical* if u is saturated under all max flows. Note that all elements of M are automatically critical.

Lemma 4.1 *Every edge has a critical endpoint. There is an efficient algorithm to determine whether a given node is critical.*

Proof. Let $(u, v) \in E$. If neither u nor v is critical, then there exist max flows f, g such that f does not saturate u and g does not saturate v. By linearity, any non-extremal convex combination of f and g, say $(f + g)/2$, would be a max flow that saturates neither u nor v. This is a contradiction, since one could increase the flow along the edge (u, v).

To determine whether u is critical, we can solve the max flow problem with the capacity of u reduced to $c(u) - 1$. The node u is critical iff either the resulting flow problem is not feasible or the max flow value is strictly less. □

Call an edge e *useless* if $f(e) = 0$ for all max flows f.

Lemma 4.2 *There is an efficient algorithm to determine whether a given edge is useless.*

Proof. Place a small nonzero lower capacity bound on the edge e and solve for a max flow. The edge e is useless iff either the resulting flow problem is not feasible or the max flow value is strictly less. □

A more efficient algorithm is given by the following lemma, which allows all the useless edges to be found with a single max flow computation plus an additional linear-time computation.

Lemma 4.3 *Let f be a max flow in (U, V, E, c, M). Form the residual graph $G_f = (U, V, E_f)$, in which $(u, v) \in E_f$ if $(u, v) \in E$ or if $(v, u) \in E$ and $f(v, u) > 0$. Find the strongly connected components of G_f. Then*

(i) *any edge between two nodes in the same strongly connected component is not useless;*

(ii) *any edge on a path in G_f from an unanchored node of V to an unsaturated node of V is not useless;*

(iii) *any edge on a path in G_f from an unsaturated node of U to an unanchored node of U is not useless;*

(iv) *all other edges are useless.*

Proof. Note that we only need to consider edges $e \in E$ such that $f(e) = 0$, since any edge with nonzero flow falls in case (i).

For (i), there is a directed cycle in G_f through any edge between two nodes in the same strongly connected component. A small nonzero amount of flow can be circulated around the cycle without changing the flow value or violating any constraints, ensuring that all edges on the cycle have nonzero flow.

For (ii), if there is a directed path from an unanchored node of V to an unsaturated node of V, a small amount of flow can be routed along that path without changing the flow value or violating any constraints, ensuring that all edges along the path have nonzero flow. The argument for (iii) is symmetric.

Now we show that all other edges are useless. Add the following edges to the graph:

(a) from each unsaturated node of V to each unsaturated node of U;

(b) from each unsaturated node of V to each unanchored node of V;

(c) from each unanchored node of U to each unsaturated node of U.

These extra edges may create new cycles, but we argue that any edge in E contained in a cycle that was not previously contained in a cycle satisfies either (ii) or (iii), therefore is not useless. First, consider the circular sequence of new edges (those of type (a), (b), or (c)) in order around a new cycle. The sequence cannot contain adjacent pairs of type (a)-(a), (a)-(b), (c)-(a), or (c)-(b), otherwise there would be a path in G_f from an unsaturated node of U to an unsaturated node of V. This would be an augmenting path, contradicting the maximality of $|f|$. It follows that there can be no new cycle containing an edge of type (a) and no new cycle containing both an edge of type (b) and an edge of type (c). Thus the only new cycles contain edges of type (b) or (c), but not both. If the cycle contains edges of type (b), then any edge in E on the cycle lies on a path in G_f from an unanchored node of V to an unsaturated node of V, therefore satisfies (ii). Similarly, if the cycle contains edges of type (c), then any edge in E on the cycle satisfies (iii).

Now form the strongly connected components of this graph and topologically sort the quotient graph to obtain a linear order of the strongly connected components. (Note that this is not part of the algorithm, but only part of the proof of the lemma.) Let X be a prefix of the sorted list of components and Y the complementary suffix. Any edge of E not contained in a cycle crosses some such cut X, Y, and all edges between X and Y go from X to Y and have no flow under f. There are three cases to consider:

1. All unsaturated elements of V are in X and all unsaturated elements of U are in Y.
2. X contains an unsaturated element of U.
3. Y contains an unsaturated element of V.

There is no other possibility due to the edges (a) above.

In case 1, all elements of $X \cap U$ and all elements of $Y \cap V$ are saturated. In this case,

$$\begin{aligned} |f| &= \sum_{u \in X \cap U} r_f(u) \; + \sum_{v \in Y \cap V} r_f(v) \; - \sum_{e \in E \cap X \times Y} f(e) \\ &= \sum_{u \in X \cap U} c(u) \; + \sum_{v \in Y \cap V} c(v) \; - \; 0, \end{aligned}$$

and if g is any other max flow,

$$\begin{aligned} |f| = |g| &= \sum_{u \in X \cap U} r_g(u) \; + \sum_{v \in Y \cap V} r_g(v) \; - \sum_{e \in E \cap X \times Y} g(e) \\ &\leq \sum_{u \in X \cap U} c(u) \; + \sum_{v \in Y \cap V} c(v) \; - \sum_{e \in E \cap X \times Y} g(e), \end{aligned}$$

therefore

$$\sum_{e \in E \cap X \times Y} g(e) = 0.$$

Since g was an arbitrary max flow, all edges in $E \cap X \times Y$ are useless.

In case 2, all elements of $Y \cap V$ are saturated due to edges (a), and all elements of $Y \cap U$ are anchored due to edges (c). In this case,

$$\sum_{v \in Y \cap V} c(v) \; = \sum_{v \in Y \cap V} r_f(v) \; = \sum_{u \in Y \cap U} r_f(u) \; + \sum_{e \in E \cap X \times Y} f(e) \; = \sum_{u \in Y \cap U} c(u) + 0.$$

If g is any other max flow,

$$\begin{aligned} \sum_{v \in Y \cap V} c(v) \; + \sum_{e \in E \cap X \times Y} g(e) &= \sum_{u \in Y \cap U} c(u) \; + \sum_{e \in E \cap X \times Y} g(e) \\ &= \sum_{u \in Y \cap U} r_g(u) \; + \sum_{e \in E \cap X \times Y} g(e) \\ &= \sum_{v \in Y \cap V} r_g(v) \\ &\leq \sum_{v \in Y \cap V} c(v), \end{aligned}$$

therefore

$$\sum_{e \in E \cap X \times Y} g(e) = 0.$$

Since g was an arbitrary max flow, all edges in $E \cap X \times Y$ are useless.

Case 3 and case 2 are symmetric. □

The lemma allows the useless edges to be found in linear time plus the time to find a max flow.

4.3 Finding a Min Cut

Let (U, V, E, c, M) be a flow problem. A *connected component* is an equivalence class of the equivalence relation on $U \cup V$ generated by E. Two nodes are in the same connected component if they are connected by a zigzag path in $(E \cup E^{-1})^*$.

Lemma 4.4 *Let (U, V, E, c, M) be a flow problem with no useless edges. Let C be a connected component. Either all elements of $C \cap U$ are critical or all elements of $C \cap V$ are critical. There is a linear-time algorithm to determine which.*

Proof. Suppose for a contradiction that both U and V contain noncritical elements of C. Then there exist noncritical elements $u \in C \cap U$ and $v \in C \cap V$ and a zigzag $E \cup E^{-1}$-path p from u to v such that all intermediate nodes on the path are critical. Since u and v are noncritical, there is a max flow that does not saturate u and one that does not saturate v. Moreover, since there are no useless edges, for each edge on the path p, there is a max flow assigning nonzero flow to that edge. By linearity, any non-extremal convex combination of these max flows is still a max flow, but has p as an augmenting path. This is a contradiction.

If the total capacity of $C \cap U$ is less than that of $C \cap V$, then all elements of $C \cap U$ are critical and not all those of $C \cap V$. Similarly, if the total capacity of $C \cap V$ is less than that of $C \cap U$, then all elements of $C \cap V$ are critical and not all those of $C \cap U$. All elements of C are critical iff the total capacities of $C \cap U$ and $C \cap V$ are equal. □

Let (U, V, E, c, M) be a feasible flow problem with no useless edges. Let M' be a set of critical nodes defined as follows. For each connected component C, include $C \cap U$ in M' if all nodes of $C \cap U$ are critical, otherwise include $C \cap V$ in M'. All nodes of M' are critical by Lemma 4.4. (If all nodes of C are critical, we might just as well have chosen to include $C \cap V$ in M' instead of $C \cap U$; the choice is arbitrary.) The set M' can be found efficiently by comparing the total capacities of $C \cap U$ and $C \cap V$, as described in the proof of Lemma 4.4.

Lemma 4.5 *Let (U, V, E, c, M) be a feasible flow problem. Let $m = \max |f|$ be the max flow value. The set M' defined above is a min cut in the sense that*

$$m = \sum_{u \in M'} c(u).$$

Proof. The inequality $\leq$ holds because every edge has exactly one endpoint in M', thus every unit of flow passes through exactly one element of M'. Equality holds for max flows because every element of M' is critical, therefore saturated. □

Lemma 4.6 *Let (U, V, E, c, M) be a feasible flow problem and let $M' \subseteq U \cup V$ be the min cut of Lemma 4.5. The flow problem $(U, V, E, c, M \cup M')$ is feasible, and every flow is a max flow.*

Proof. A max flow in (U, V, E, c, M) is a flow in $(U, V, E, c, M \cup M')$, and the matching lower and upper bounds on M' force the flow value to match the capacity of the cut. □

4.4 An Algorithm

Let (U, V, E, c, M) be a flow problem. Let $B_i = A_1 \cup \cdots \cup A_i$. Define a sequence of flows f_i, an increasing sequence of anchor sets $M_i \supseteq M$, and a decreasing sequence of edge sets $E_i \subseteq E$, $0 \leq i \leq n$, inductively as follows. The invariants of the algorithm are

(i) f_i is a flow in $(U, V, E_i \cap B_i, c, M_i)$ (thus also in (U, V, E_i, c, M_i));

(ii) all flows in (U, V, E_i, c, M_i) are lexicographically equivalent with respect to $A_1, \ldots, A_i$; that is,

$$f \restriction B_i \equiv_{\text{lex}} g \restriction B_i$$

for any flows f, g in (U, V, E_i, c, M_i);

(iii) all flows in $(U, V, E, c, \varnothing)$ that are lexicographically maximum with respect to $A_1, \ldots, A_i$ are flows in (U, V, E_i, c, M_i).

Initially, $f_0 = 0$, $M_0 = \varnothing$, and $E_0 = E$.

Now suppose we have computed f_{i-1}, M_{i-1}, and E_{i-1} satisfying the invariants. Let $m \stackrel{\text{def}}{=} |f_{i-1} \restriction B_{i-1}|$. By invariant (ii), all flows g in $(U, V, E_{i-1}, c, M_{i-1})$ satisfy $|g \restriction B_{i-1}| = m$.

Consider the flow problem $(U, V, E_{i-1} \cap B_i, c, M_{i-1})$. This problem is feasible, since f_{i-1} satisfies invariant (i) and $E_{i-1} \cap B_{i-1} \subseteq E_{i-1} \cap B_i$. Find a max flow f_i. Then $f_i \restriction B_{i-1}$ is lexicographically maximum in $(U, V, E_{i-1} \cap B_{i-1}, c, M_{i-1})$, and $|f_i \restriction i| = |f_i| - |f_i \restriction B_{i-1}| = |f_i| - m$ is maximum among all flows in $(U, V, E_{i-1} \cap B_i, c, M_{i-1})$, therefore f_i is lexicographically maximum in $(U, V, E_{i-1} \cap B_i, c, M_{i-1})$.

Let M_i be the new set of anchors as described in Lemma 4.5, and let E_i consist of E_{i-1} less the edges in $A_{i+1} \cup \cdots \cup A_n$ that have an endpoint in M_i. These definitions force any flow in (U, V, E_i, c, M_i) to be lexicographically equivalent with respect to $A_1, \ldots, A_i$ to f_i, which reestablishes invariant (ii). Moreover, no flow in $(U, V, E, c, \varnothing)$ that is lexicographically maximum with respect to $A_1, \ldots, A_i$ can use edges in $E_{i-1} - E_i$, thus invariant (iii) is reestablished.

We have shown

Theorem 4.7 *The flow f_n is a lexicographically maximum flow in (U, V, E, c, M).*

Acknowledgements

Many thanks to Saleh Elmohamed, Bobby Kleinberg, Jon Kleinberg, Alexa Sharp, Dan Sheldon, Éva Tardos, and David Williamson. This work was supported by NSF grant CCF-0635028.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice Hall, 1993.

[2] Vašek Chvátal. *Linear Programming*. W. H. Freeman, 1983.

[3] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley Interscience, 1998.

[4] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2nd edition, 2002.

[5] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 1st edition, 2005.

[6] N. Megiddo. A good algorithm for lexicographically optimal flows in multiterminal networks. *Bull. AMS*, 83(3):407–409, 1977.