(Almost) Tight Bounds and Existence Theorems for Single-Commodity Confluent Flows

Jiangzhuo Chen [*] Robert D. Kleinberg [†] László Lovász [‡]
Rajmohan Rajaraman [*] Ravi Sundaram [§] Adrian Vetta [¶]

Abstract

A flow of a commodity is said to be confluent if at any node all the flow of the commodity leaves along a single edge. In this paper we study single-commodity confluent flow problems, where we need to route given node demands to a single destination using a confluent flow. Single- and multi-commodity confluent flows arise in a variety of application areas, most notably in networking; in fact, most flows in the Internet are (multi-commodity) confluent flows since Internet routing is destination based.

We present near-tight approximation algorithms, hardness results, and existence theorems for minimizing congestion in single-commodity confluent flows. The maximum edge congestion of a single-commodity confluent flow occurs at one of the incoming edges of the destination. Therefore, finding a minimum-congestion confluent flow is equivalent to the following problem: given a directed graph G with k *sinks* and non-negative demands on all the nodes of G, determine a confluent flow that routes every node demand to some sink such that the maximum congestion at a sink is minimized [2].

The main result of this paper is a polynomial-time algorithm for determining a confluent flow with congestion at most $1 + \ln(k)$ in G, if G admits a splittable flow with congestion at most 1. We complement this result in two directions. First, we present a graph G that admits a splittable flow with congestion at most 1, yet no confluent flow with congestion smaller than H_k, the k^{th} harmonic number, thus establishing tight upper and lower bounds to within an additive constant less than 1. Second, we show that it is NP-hard to approximate the congestion of an optimal confluent flow to within a factor of $(\log_2 k)/2$, thus resolving the polynomial-time approximability to within a multiplicative constant. We also consider a demand maximization version of the problem. We show that if G admits a splittable flow of congestion at most 1, then a variant of the congestion minimization algorithm yields a confluent flow in G with congestion at most 1 that satisfies $1/3$ fraction of total demand.

We show that the gap between confluent flows and splittable flows is much smaller, if the underlying graph is k-connected. In particular, we prove that k-connected graphs with k sinks admit confluent flows of congestion less than $C + d_{\max}$, where C is the congestion of the best splittable flow, and $d_{\max}$ is the maximum demand of any node in G. The proof of this existence theorem is non-constructive and relies on topological techniques introduced by Lovász [19].

[*]College of Computer & Information Science, Northeastern University, Boston, MA 02115. Email: {chenj,rraj}@ccs.neu.edu. Partially supported by NSF CAREER award CCR-9983901.

[†]MIT Department of Mathematics, Cambridge, MA 02139, Email: rdk@math.mit.edu. Supported by a Fannie and John Hertz Foundation Fellowship.

[‡]Microsoft Research, One Microsoft Way, Redmond, WA 98052. Email: lovasz@microsoft.com.

[§]College of Computer & Information Science, Northeastern University, Boston, MA 02115. Email: koods@ccs.neu.edu. Part of this work was done while the author was at Akamai Technologies, Cambridge, MA.

[¶]Department of Mathematics and School of Computer Science, McGill University, Montreal, Quebec, H3A 2A7, Canada. Email: vetta@math.mcgill.ca. Part of this work was done while the author was attending an algorithms workshop, supported by an ONR basic research grant, at Bell Labs.

1 Introduction

In this paper, we present new approximation algorithms, lower bounds, and existence theorems for a class of network flows called *confluent flows*. A flow of a given commodity in a directed graph is said to be *confluent* if all the flow of the commodity departing a node does so along a single outgoing edge.

Confluent flows arise in a number of scenarios including evacuation problems and various applications in networking. For instance, content delivery networks (CDNs) often organize their deployment of servers in the form of a rooted tree with each node forwarding data from its children to its parent (e.g., log data, transactional data or forms) and vice versa (e.g., html data or streaming data from the content provider). In the preceding example, the flow resulting from the collection of log and transactional data is confluent. The wireless domain provides another example where ad hoc networks of Wi-Fi access points act as forwarding agents back to the wired access point that connects to the Internet.

Perhaps the most common application of confluent flows is in Internet routing. Most flows on the Internet today are confluent because Internet routing is primarily based on selecting a shortest path tree to each destination and then routing along the selected shortest paths. The packet flow for each destination is confluent since all packets departing a router for a particular destination depart along the same edge. A major shortcoming of shortest-paths routing, however, is that it ignores congestion at intermediate nodes and edges since the shortest paths are usually calculated independently for each source-destination pair. This raises the following natural network flow problem: Given a directed graph, a set of source and destination nodes, a capacity for each edge, and a demand for each source-destination pair, determine a minimum-congestion flow that routes all the demand and is confluent for each destination. Here, we define the congestion of an edge to be the ratio of the flow on the edge to its capacity, and the congestion of a flow is the maximum congestion, among all edges.

The above confluent flow problem was posed in recent work [2], where it was shown that minimizing confluent flow congestion is MAXSNP-hard even for the single commodity (destination) case with uniform node demands and uniform edge capacities, and that an $\widetilde{O}(\sqrt{n})$ approximation is achievable for an n-node graph for this special case. For the multi-commodity case, where we need to obtain confluent flows for multiple destinations, it was shown that a logarithmic-approximation is achievable if the capacity of every edge is at least as large as the total demand for any destination [2].

In this paper, we present near-tight bounds on the approximability of single-commodity confluent flows in networks with arbitrary non-uniform demands and uniform edge capacities. We also study the gap between the congestion of an optimal confluent flow and that of an optimal splittable flow (which need not obey the confluence constraint). The more realistic confluent flow problems that incorporate multiple commodities (destinations) and arbitrary edge capacities appear to be beyond the reach of our present algorithmic techniques. We leave these as open questions for further research that can hopefully capitalize on the techniques we develop in this paper.

1.1 Our results

Consider the single-commodity confluent flow problem with uniform edge capacities. In this case, the congestion of an edge is simply the flow going through the edge. Therefore, the maximum edge congestion of any confluent flow is identical to the maximum flow among the nodes that have an edge into the destination [2]. Hence, we consider an equivalent problem where the graph is transformed by removing the destination and designating its incoming neighbors as sinks.

Formally, consider a directed graph G with k distinguished nodes, referred to as *sinks*, and non-negative demands on all the nodes of G. In the *single-commodity multi-sink confluent flow problem*, we seek a confluent flow that routes every node demand to some sink such that the maximum flow arriving at any sink, referred to as the *congestion* of the sink, is minimized. In addition to being a useful reformulation of the original single-commodity confluent flow problem, the multi-sink version generalizes a classic graph partitioning question studied more than three decades ago, as we discuss below, and is thus of independent interest. The remainder of this paper will focus on the single-commodity multi-sink confluent flow problem.

- Our main result is a polynomial-time algorithm for determining a confluent flow with congestion at most $1+\ln(k)$ in G, if G admits a splittable flow with congestion at most 1 (Section 4). We complement this result by presenting a graph G that admits a splittable flow with congestion at most 1, yet no confluent flow with congestion smaller than H_k, the k^{th} harmonic number (Section 3.1). Since $H_k = \ln k + \gamma - o(1)$, where γ is Euler's constant, we have resolved the gap between confluence and splittability to within an additive constant less than 1.

Our algorithm is based on a novel deterministic rounding of an optimal splittable flow, that repeatedly refines the flow by removing carefully selected edges and aggregating nodes into sinks, leading to the desired confluent flow. It is interesting to contrast the near-optimal bound achieved by our rounding scheme with the $\Omega(n^{1/4})$ bound achieved by a natural randomized rounding scheme, that selects for each node an outgoing edge with probability proportional to the flow on the edge in the splittable solution [2].

Since the optimal splittable flow congestion is a lower bound on the optimal confluent flow congestion, our algorithm achieves a $(1 + \ln k)$-*approximation* for minimizing congestion. One may ask whether an improved approximation can be achieved efficiently.

- We show that it is NP-hard to approximate the congestion of an optimal confluent flow to within a factor of $(\lg k)/2$,[1] thus resolving the polynomial-time approximability to within a multiplicative constant (Section 3.2). It is interesting to note that our lower bound is not based upon a reduction from set cover [4] and relies on a weaker precondition than that used in the set cover hardness result.

While the bound of $1+\ln(k)$ on the ratio between the congestion of confluent and splittable flows is existentially tight up to an additive constant, it is natural to wonder if there are interesting classes of graphs for which the gap is smaller.

- A positive answer to this question is provided in Section 6, in which we prove that k-connected graphs with k sinks admit confluent flows of congestion less than $C + d_{\max}$, where C is the congestion of the best splittable flow and $d_{\max}$ is the maximum demand of any node. In particular, this means that the ratio between confluent and splittable congestion in k-connected graphs is at most 2. Interestingly, the proof of this existence theorem is non-constructive and relies on topological techniques introduced in [19].

Finally, we also consider a demand maximization problem, in which we seek a confluent flow with congestion at most 1 that maximizes the total demand of all the nodes whose demand is satisfied.

- We show that if G admits a splittable flow of congestion at most 1, then a variant of the congestion minimization algorithm yields a confluent flow in G with congestion at most 1 that satisfies 1/3 fraction of total demand (Section 5). We also show that the demand-maximization problem is NP-hard to approximate to a factor less than 4/3 (Section 3.2).

[1] Throughout this paper, we use lg to refer to $\log_2$.

Our 3-approximation result for demand maximization assumes the existence of a splittable flow that satisfies all demands with congestion at most 1. We can eliminate this assumption at the expense of increasing the approximation ratio by a constant factor by first determining a splittable flow of congestion at most 1 that satisfies a constant fraction of the maximum satisfiable demand; any of the algorithms of [3, 15] for single-source unsplittable flow would suffice for this purpose.

1.2 Related work

The bulk of our results in this paper compare confluent flows with a natural relaxation, namely splittable flows, which are well-characterized by the celebrated max-flow min-cut theorem of Ford and Fulkerson [5, 6]; there is a vast literature on efficient algorithms for obtaining the maximum flow. Another relaxation of confluent flow is unsplittable flow, which requires that the demand for every source be routed along a single path. Both the congestion minimization and demand maximization versions of unsplittable flow may be approximated to within a constant factor using the algorithms of [3, 15]. The relationship between the *edge congestion* of confluent and unsplittable flows is addressed in [18], in which an $\Omega(n)$ separation is established, where n is the number of nodes.

In the special case where G is an undirected graph and all vertices have unit demand, finding a confluent flow of congestion $\leq C$ is equivalent to partitioning G into k distinct connected subgraphs of size $\leq C$, each containing one of the sinks. (Given such a partition, a confluent flow is obtained by routing all flow along the edges of a spanning tree in each subgraph.) When G is k-vertex-connected, Frank [8] conjectured in 1975 that such partitions always exist, provided that $kC \geq n$. In fact, he made the much stronger conjecture that given sinks $s_1, \dots, s_k$ and positive integers $n_1, \dots, n_k$ summing to n, one could partition G into k connected subgraphs, such that the i-th subgraph contains s_i and has *exactly* n_i vertices. This conjecture was proved independently by Lovász [19] and Győry [12]. Lovász's proof applies also to directed graphs. Our result on the existence of confluent flows in k-connected graphs can be viewed as a weighted generalization of this theorem, in which the vertices are given non-negative real weights (demands) and one seeks to partition G into connected subgraphs whose total weights approximate a specified k-tuple of target weights.

In this paper, we have entirely focused on single commodity confluent flows. Multicommodity and fractional variants of confluent flows are studied in [2]. Multicommodity confluent flows are considered by [10, 16] in a model where the demands are not associated with individual source-sink pairs; instead with sources or sinks, as a whole. Also related is the work of [14], which raises the problem of finding a subtree of a given network that can route a given set of multicommodity flow pairs with minimum congestion. The impact of confluence on IP routing is studied in [18] and [21].

2 Confluent flow problem definitions

Let $G = (V, E)$ denote a directed graph and $d : V \to \mathbb{R}_+$ denote a function specifying the demand, $d(v)$, at each vertex, v. For a set $X \subseteq V$ we define $d(X) = \sum_{v \in X} d(V)$. We denote the total demand, $d(V)$, by D and the maximum demand, $\max_{v \in V} d(v)$, by $d_{\max}$. Let $S = \{s_1, \dots, s_k\} \subseteq V$ denote a set of k sinks. Any flow $f : E \to \mathbb{R}_+$ routing the demands to the sinks satisfies the flow conservation equation

$$\sum_{e=(v,w)\in E} f(e) - \sum_{e=(u,v)\in E} f(e) = d(v)$$

at every vertex $v \in V - S$. For any node v, define the *in-flow* of v, $\text{in}(v)$, to be the sum of the flows on the edges into v. The congestion of f at a node v is now defined as $d(v) + \text{in}(v)$. The congestion of f is defined as the maximum, over all nodes v, of the congestion of f at v.

Without loss of generality, we assume that each sink has only incoming edges. (Supposing s_i has outgoing edges, then we can add a sink vertex s'_i and an edge (s_i, s'_i) to G and remove s_i from S.)

We say that a flow f is confluent if for every node u, there exists at most one edge (u, v) that has positive flow (i.e., $f(u, v) > 0$). Thus, a confluent flow yields a subgraph of G consisting of disjoint components $\{T_1, \ldots, T_k\}$, such that each T_i is an arborescence directed towards the root s_i. In any arborescence T_i, the maximum node congestion occurs at the sink s_i and equals the total demand in T_i, given by $d(T_i)$. We refer to the maximum node congestion $\max_i d(T_i)$ as the congestion of the confluent flow.

In this paper, we consider two optimization problems concerning confluent flows. In the *congestion minimization* problem, we seek a confluent flow with minimum congestion among all confluent flows that satisfy all demands. In the *demand maximization* problem, we seek a confluent flow with congestion at most 1 that maximizes the total demand of all the nodes whose demand is satisfied.

Our confluent flow algorithms are based on a deterministic rounding of a splittable flow with minimum congestion. In the following, we review polynomial-time algorithms for computing such a flow.

Review of algorithms for computing a congestion-minimizing splittable flow. A splittable flow that minimizes congestion can be obtained in polynomial time by a binary search procedure, that searches for the minimum congestion by repeatedly invoking a standard maximum flow algorithm to determine whether it is feasible to route all demands given a congestion value. The preceding procedure, however, is not strongly polynomial since the number of iterations of the binary search may depend on the demand values. A splittable flow with minimum congestion can, in fact, be found in strongly polynomial time by reducing to the *perfect sharing problem*, which can be solved using standard *parametric maximum flow* techniques [9]. For the sake of completeness we provide a brief description in the following.

We consider a one-parameter family of capacitated networks with a common underlying graph structure, whose edge capacities are functions of a real-valued parameter t. Assuming that edge capacity is a non-decreasing linear function of t for edges leaving the source, a non-increasing linear function of t for edges entering the sink, and constant for all other edges, it can be shown that the min-cut capacity is a piecewise linear concave function of t with at most $n - 2$ breakpoints, and can be found in $O(nm \log(n^2/m))$ time, where n is the number of nodes and m is the number of edges in the network [9]. The preceding result yields a strongly polynomial time solution to the perfect sharing problem [9], which we now describe and show how our problem (of finding an optimal splittable flow with minimum congestion) can be reduced to it.

In the perfect sharing problem, we are given a directed graph with source s and sink t. Let $S = \{\sigma_1, \cdots, \sigma_k\}$ denote the neighbors of s, with edges (s, σ_i) having infinite capacity for each $i = 1, \cdots, k$. Each σ_i is assigned a positive weight w_i. Let u_i denote the flow through edge (s, σ_i). The goal of the perfect sharing problem is to find a flow with maximum value, among all flows in which u_i/w_i is identical for all $i = 1, \cdots, k$. The reduction of our problem to the perfect sharing problem is as follows. Given a directed graph $G = (V, E)$, node demand function d and set S of sinks, construct a directed graph G' which has a pair of nodes v', v'' and an edge (v', v'') for each node $v \in V$, an edge (u'', v') for each edge $(u, v) \in E$, a new source s and a new sink t, an edge (s, v') for each $v \in V$ with $d(v) > 0$, an edge (v'', t) for each $v \in S$. Set the capacity of each edge (s, v') to be infinity and that of every other edge to be 1. Set the weight of each neighbor v of s to be $d(v)$. It is easy to see that our minimum node congestion problem instance is equivalent to the perfect sharing problem instance on G'. In particular, if λ^* equals the ratio of flow to weight for any neighbor of the source s in the optimal perfect sharing flow, then the congestion of the optimal splittable flow of G is $1/\lambda^*$. Furthermore, the desired flow through any edge (u, v) in G is simply $1/\lambda^*$ times the flow in the corresponding edge (u'', v') in G'. We thus have a strongly polynomial time algorithm for finding a splittable flow with minimum congestion.

3 Lower bounds

In this section, we present three lower bound results. We first present an instance where the congestion of the optimal confluent flow is at least H_k times that of the optimal splittable flow. We then show that it is NP-hard to approximate the minimum congestion confluent flow to within a factor of $\frac{1}{2}\lg k$ and the demand maximization problem to a factor less than 2.

3.1 Confluent to splittable: H_k gap

Figure 1 shows an instance of a graph G which admits a splittable flow with congestion 1, but does not admit a confluent flow with congestion less than H_k. The vertex set of G consists of $(k+1)k/2$ sources (round nodes) represented by all ordered pairs of integers (i, j) with $1 \leq i \leq j \leq k$, and k sinks (square nodes). There are edges from round node (i, j) to round nodes $(i, j+1)$ and $(i+1, j+1)$. In addition there are edges from each of the k round nodes (i, k) to a corresponding square node. The demand at (i, j) is equal to $1/j$.

The congestion of any confluent flow in G is at least H_k, since the demand which originates at $(1, 1)$ must flow along a path of length k, through vertices whose demands are $1, 1/2, 1/3, \ldots, 1/k$, respectively. All of the demand originating at the vertices of this path must drain into the sink at the end of the path, which will therefore have congestion at least H_k. On the other hand, congestion 1 is achieved by the splittable flow which sends $\frac{j+1-i}{j+1}$ units of flow from (i, j) to $(i, j+1)$ and $\frac{i}{j+1}$ units of flow from (i, j) to $(i+1, j+1)$. We now verify that this is a valid flow with congestion 1. The in-flow at a node (i, j), where $i + j > 1$, equals $\frac{j-i}{j} + \frac{i-1}{j}$, which equals $\frac{j-1}{j}$. The demand at any node (i, j) is $1/j$. The out-flow at a node (i, j), where $j < k$, equals $\frac{j+1-i}{j+1} + \frac{i}{j+1}$, which is 1. Therefore, flow conservation holds at each node other than the sinks, and the congestion at each node is exactly 1.

3.2 Hardness of approximation

In [2] it was shown that the minimum congestion problem is NP-hard to approximate better than $\frac{4}{3}$ and hence MAXSNP-hard. Here, we refine the approach of [2] to improve the lower bound and show that it is NP-hard to approximate better than $(\lg k)/2$.

We present a hardness result for directed graphs with non-uniform demands. It is easy to modify this result to the case of uniform demands, where for each vertex v we wish to route exactly one unit of flow to a sink. Take a directed graph G with special vertices s_1, s_2, t_1, t_2. It is known [7] that it is NP-hard to determine whether or not there are vertex-disjoint directed paths, or dipaths, in G from s_1 to t_1 and from s_2 to t_2. We show that any approximation algorithm for the confluent flow problem with performance guarantee better than $\frac{1}{2}\lg k$ can be used to determine in polynomial time whether or not such disjoint dipaths exist in G. This will give our result. We remark that the gadgets we use were first applied in [11] for the edge-disjoint path problem.

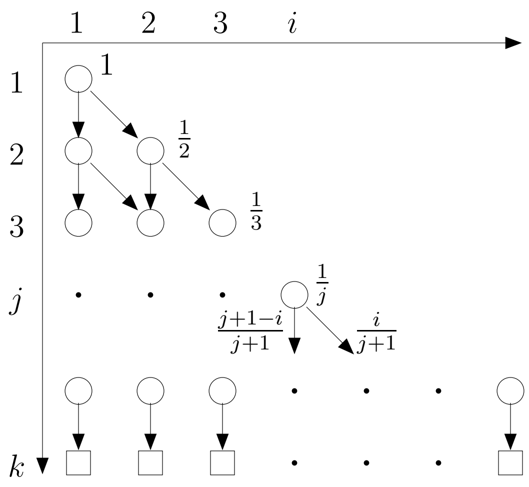

Figure 1: Instance demonstrating an H_k gap between confluent and splittable flows. The round nodes are sources with j nodes at level j each with demand $\frac{1}{j}$. The square nodes are sinks. The flows on the outgoing edges of the ith node on level j are specified.

Theorem 1 *It is NP-hard to approximate the optimal confluent flow congestion to a factor less than $\frac{1}{2}\lg k$, where k is the number of sinks.*

Proof. Given G we build an auxiliary network N as follows. Take a complete binary tree T on $2^{\lceil \lg k \rceil} - 1$ nodes, with root node r. We make T directed by replacing each edge with an arc directed away from the root. Then we replace each node v in the tree by a copy of G. We use the notation s_1^v, for example, to refer to the copy of s_1 in the copy of G associated to the node $v \in T$.

For a non-leaf node v in the tree, let $c_l(v)$ and $c_r(v)$ be its two children. Then the arc $(v, c_l(v))$ in T is replaced by the arc $(t_2^v, s_1^{c_l(v)})$ in the auxiliary network; similarly, the arc $(v, c_r(v))$ in T is replaced by the arc $(t_1^v, s_1^{c_r(v)})$ in the auxiliary network. For each leaf node u in the tree, we add the two arcs (t_1^u, t_1^{*u}) and (t_2^u, t_2^{*u}), where t_1^{*u} and t_2^{*u} are sinks. Our construction is illustrated in Figure 2.

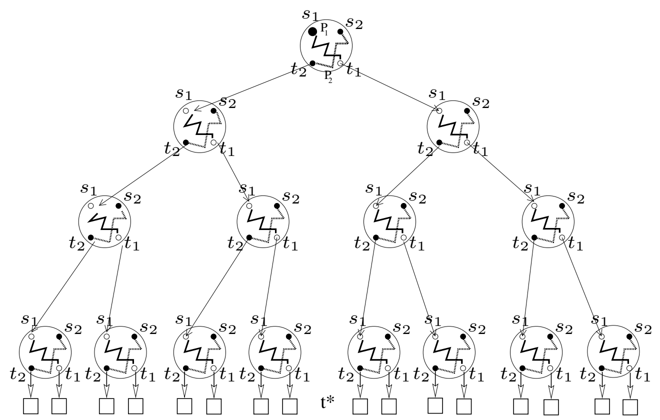

Figure 2: Instance used in the NP-hardness proof. The black vertices have demand one, except for the large black vertex s_1^r which has demand two, and the white vertices have demand zero.

In addition, we give each vertex in the auxiliary network a demand. Every copy of s_2 and t_2 has demand one. Every copy of s_1 and t_1 has demand zero, except for s_1^r which has demand two. Every other vertex has demand zero.

Now, suppose G contains vertex-disjoint dipaths P_1 and P_2 from s_1 to t_1 and from s_2 to t_2, respectively. Utilizing these dipaths in each copy of G we obtain a collection of disjoint dipaths that end at the set of sinks t^* and that cover every copy of s_1, s_2, t_1 and t_2. This is shown in Figure 2. Each dipath from a copy of s_2 to a sink contains exactly two nodes with non-zero demand: a copy of s_2 and a copy of t_2. Furthermore, the dipath from s_1^r to a sink contains only one node with non-zero demand, s_1^r. Therefore, the congestion of the resulting confluent flow is exactly two. This is optimal since there is a vertex with demand two.

Now suppose that G does not contain vertex-disjoint dipaths P_1 and P_2 from s_1 to t_1 and from s_2 to t_2. Take any confluent flow in our network and consider the dipath P it induces from s_1^r to t^*. We now show that this dipath must have congestion at least $\lceil \lg k \rceil + 1$. Towards this end, assume that P passes through copies of G corresponding to the nodes $r = v_1, v_2, \ldots, v_{\lceil \lg k \rceil}$ of T. Thus, by construction, P must pass through $s_1^{v_i}$, for $1 \le i \le \lceil \lg k \rceil$. We claim that vertex $s_1^{v_i}$ has congestion at least $i+1$. This is true for $s_1^{v_1}$. Assume then that $s_1^{v_j}$ has congestion at least $j+1$, and consider the copy of G corresponding to v_j. We know that any flow at $s_1^{v_j}$ and at $s_2^{v_j}$ must be routed via $t_1^{v_j}$ or $t_2^{v_j}$. However, because there are no disjoint dipaths P_1 and P_2, we must have one of the situations shown in Figure 3. Either the flow at $s_1^{v_j}$ is routed via $t_2^{v_j}$ and the flow at $s_2^{v_j}$ is routed via $t_1^{v_j}$, or the flows at $s_1^{v_j}$ and at $s_2^{v_j}$ are both routed via the same vertex (either $t_1^{v_j}$ or $t_2^{v_j}$). In each case, the congestion at either $t_1^{v_j}$ or $t_2^{v_j}$ is at least $(j+1)+1$. This congestion is passed down to vertex $s_1^{v_{j+1}}$. Thus, by induction, it follows that the congestion of the vertex $s_1^{v_{\lceil \lg k \rceil}}$ is $\lceil \lg k \rceil + 1$. Observe that the number of sinks in the auxiliary network is $2^{\lceil \lg k \rceil} = k$, if k is a power of 2.

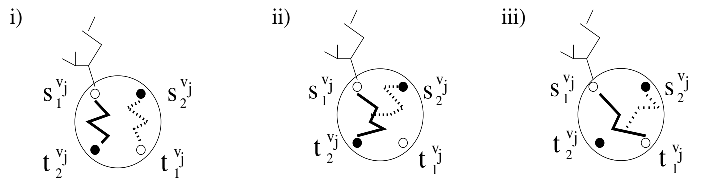

Figure 3: Routing in Networks without Disjoint Dipaths.

It follows that any approximation algorithm for the confluent flow problem with approximability guarantee better than $\frac{1}{2} \lg k$ can be used to determine in polynomial time whether a directed graph contains vertex disjoint dipaths from s_1 to t_1 and from s_2 to t_2. ■

Finally, we show that the demand maximization problem is NP-hard to approximate better than 2.

Theorem 2 *It is NP-hard to approximate the confluent flow demand maximization problem to a factor less than* 2*.*

Proof: We'll use the same graph G as in the proof of Theorem 1, with demands of 1 at s_1, $1-\varepsilon$ at s_2, 0 at t_1, ε at t_2, and 0 at every other node. If G contains vertex-disjoint dipaths from s_1 to t_1 and from s_2 to t_2, we can route all the flow along these paths to achieve congestion 1. If not, then there is no way to achieve congestion 1 without excluding at least one of s_1 and s_2. Thus it is NP-hard to distinguish whether the maximum satisfiable demand is equal to 2 or is at most $1+\varepsilon$. As $\varepsilon \to 0$, this gives the bound of 2 stated in the theorem. ■

4 Congestion minimization

In this section, we present a polynomial-time algorithm to determine a confluent flow that satisfies all demands and has congestion at most $1 + \ln(k)$. We present our algorithm and its analysis in two stages. We first describe in Section 4.1 an algorithm that achieves a congestion of $1 + \lg k$. In Section 4.2, we refine the algorithm of Section 4.1 and its analysis to obtain the desired $1 + \ln k$ congestion bound.

4.1 A $(1 + \lg k)$-congestion algorithm

We begin by giving a brief overview of the algorithm. Our starting point is a (splittable) flow f in G that routes all demands to the sinks and has minimum congestion. We will assume, without loss of generality, that the congestion of this optimal splittable flow is 1, since this property can always be ensured by scaling all demands and flows by the congestion value. We may also assume, without loss of generality, that the given directed graph G is, in fact, the directed acyclic graph (dag) induced by the splittable flow.

As the algorithm proceeds, it transforms the graph G and the flow f by repeatedly performing one of three operations: (i) aggregate a node into a sink if all of the outgoing edges of that node are to the same sink; (ii) remove an edge (often by breaking certain undirected cycles[2]) and redirect flow; and (iii) deactivate a sink by removing all edges incident into the sink and redirecting flow. These operations are illustrated in Figure 4. While these operations repeatedly make changes to the graph G (edges and nodes are removed), the set S of sinks (sinks are deactivated), the flow f, and the demands d (nodes are removed), we always maintain the following invariants:

1. f always satisfies d and the flow conservation constraints.
2. The congestion at any node $v \in V(G) - S$ never increases.
3. There are only incoming edges at each sink.

At termination, the transformed graph consists of k nodes. Each node represents a set of nodes in the original graph that have been aggregated into a sink (including the sink). Any spanning forest of the edges contracted during the aggregation process yields the disjoint trees $\{T_1, \ldots, T_k\}$ forming the desired confluent flow. In the algorithm specified below, the edges of such a forest are marked during the node aggregation steps.

We are now ready to define our algorithm Conflt. At any stage of Conflt, for any sink s_i, let b_i denote the congestion of s_i (note that this is simply $d(s_i) + \text{in}(s_i)$). A node v is referred to as a *frontier node* if v has an edge incident into one of the sinks.

Conflt(G, S, d, f).

While $V(G) \neq S$, execute the following *main loop*:

1. **If** a frontier node v has all of its outgoing edges going into one sink s_i, then
   (a) **Node aggregation:** Mark any one of these edges, contract v into s_i, and add $d(v)$ to $d(s_i)$.
2. **Else**, let $\hat{G}$ denote the graph obtained from G by adding an edge $e_{\text{rev}} = (s, v)$ for each edge $e = (v, s)$ from a frontier node to a sink.

[2]We note that cycle-breaking is also an important component of the approximation algorithms for unsplittable flow [15, 3].

(a) **If** $\hat{G}$ contains a directed simple cycle C of length greater than 2, then

i. **Breaking sawtooth cycles:** Update f as follows. Let $f_{\min}$ denote the minimum flow value on any edge of $C \cap E(G)$. For each edge e of G, if $e \in C$ decrease $f(e)$ by $f_{\min}$ and if $e_{\text{rev}} \in C$ increase $f(e)$ by $f_{\min}$. Remove all edges with zero flow.

(b) **Else**

i. **Sink deactivation:** Let s_j be a sink node with only one adjacent frontier node v, and let s_ℓ be another sink node adjacent to v. If $b_j + f(v, s_\ell) < b_\ell - f(v, s_\ell)$, remove edge (v, s_ℓ) and send all its flow along edge (v, s_j); otherwise, remove edge (v, s_j), send all its flow along edge (v, s_ℓ), and deactivate s_j. (Recall that b_i denotes the congestion of sink s_i.)

EndWhile

Output: Output the marked edges.

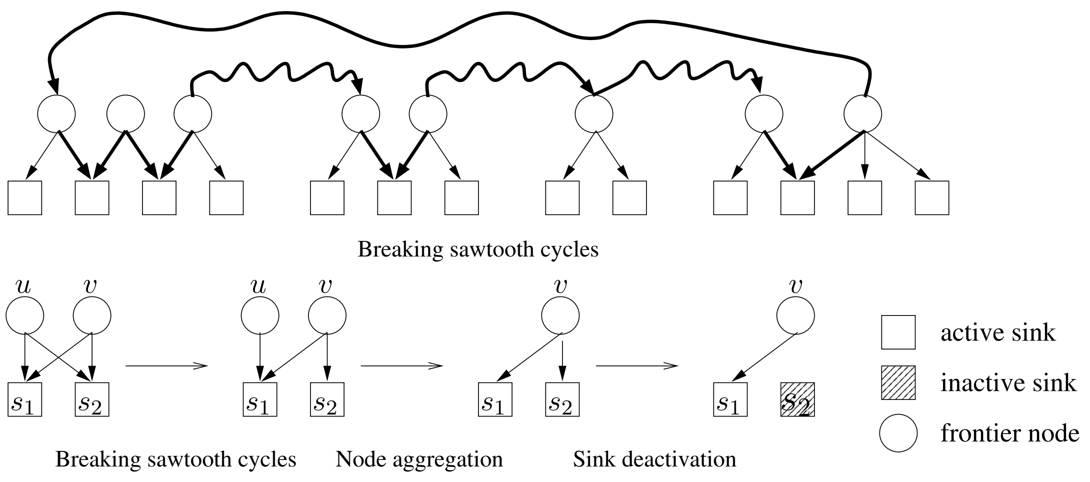

Figure 4: Illustrating steps 2(a)i, 1a, and 2(b)i of CONFLT. In the top figure, the bold edges form a sawtooth cycle.

Each node aggregation and sink deactivation step can be executed by a scan through the adjacency lists of the nodes. The step of breaking sawtooth cycles can be executed by constructing $\hat{G}$ and then performing a depth-first traversal of $\hat{G}$, both of which take time linear in the size of G. In each iteration, the size of the graph decreases, thus yielding a polynomial bound on the running time of CONFLT (a naive analysis yields an $O(m^2 + mn \log(n^2/m))$ bound).

Theorem 3 *Given a splittable flow with node congestion* 1 *on a graph with* k *sinks,* CONFLT *finds a confluent flow that satisfies all demands and has congestion at most* $1 + \lg k$.

Our proof is by a potential function argument. We define the potential of sink s_i as $\phi(s_i) = 2^{b_i}$, and the potential of the flow as the sum of the potentials of the active sinks. In the following sequence of lemmas, we show that the potential of the flow never increases.

Lemma 4 (Node aggregation) *If a frontier node has all of its outgoing edges going into one sink, then the operation in step 1a does not increase the congestion of any node, and removes at least one node.*

Proof: Aggregating the frontier node v into a sink s_i increases $d(s_i)$ by $d(v)$ but does not change b_i. The node v is removed, and the flow along all the other edges remains unchanged. ■

Lemma 5 (Breaking sawtooth cycles) *If there is a directed simple cycle C of length > 2 in $\hat{G}$, then the operation in step 2(a)i does not increase the congestion of any node, and removes at least one edge.*

Proof: At least one edge of $C \cap E(G)$ has $f_{\min}$ units of flow prior to performing the operation, and this edge is removed. To prove that the operation doesn't increase the congestion of any node, let v be any node of G. If v does not belong to C, then the flow on edges incident to v is unchanged. If v belongs to C, then C has two edges incident to v, and these are either two edges of G, or an edge of G and the reverse of another edge. In the first case, the congestion of v decreases by $f_{\min}$; in the second case, it is unchanged. ■

Lemma 6 (Sink deactivation) *If none of the preconditions of Lemmas 4 and 5 hold, then the sink deactivation step can be performed, and it either removes an edge or deactivates a sink without increasing the potential of the resulting flow. Furthermore, the potential of the deactivated sink, if any, is no more than the potential of the flow before performing the sink deactivation.*

Proof: If the preconditions of Lemmas 4 and 5 do not hold, then there is a sink node s_j which is adjacent to only one frontier node v. Indeed, we may find such a sink node by walking along edges of $\hat{G}$, always obeying the rules

1. when possible, an edge (u, v) is not followed by the reverse edge (v, u)
2. when possible subject to (1), we avoid visiting sink nodes.

Every vertex of $\hat{G}$ has at least one outgoing edge, so the walk will go on indefinitely. Let v be the first vertex visited twice. By assumption $\hat{G}$ has no simple cycles of length > 2, so the last two hops of the walk were (v, w) and (w, v) for some vertex w, and all outgoing edges of w lead to v. Either v or w must be a sink (else (v, w) and (w, v) are both in $E(G)$ contradicting the fact that G is a dag), and v cannot be a sink (as then w would satisfy the precondition of Lemma 4). Hence w is a sink s_j, and v is the only adjacent frontier node. Moreover, v is adjacent to at least one other sink s_ℓ; else v either satisfies the precondition of Lemma 4, or it has an edge to another non-sink node, in which case the walk would have traversed this edge rather than (v, w) according to rule (2).

If $b_j + f(v, s_\ell) < b_\ell - f(v, s_\ell)$, we remove (multi)edge (v, s_ℓ) and increase the flow on (v, s_j) by $f(v, s_\ell)$. It follows from the convexity of ϕ that the potential does not increase.

If $b_j + f(v, s_\ell) \geq b_\ell - f(v, s_\ell)$, we remove the (multi)edge (v, s_j), increase the flow on (v, s_ℓ) by $f(v, s_j)$, and deactivate s_j. The increase in potential is at most

$$\begin{aligned} & 2^{b_\ell + f(v,s_j)} - 2^{b_\ell} - 2^{b_j} \leq 2^{b_\ell + f(v,s_j)} - 2^{b_\ell} - 2^{b_\ell - 2f(v,s_\ell)} \\ \leq\ & 2^{b_\ell + f(v,s_j)} - 2^{b_\ell - f(v,s_\ell) + 1} \leq 0. \end{aligned}$$

(The second step follows from the convexity of the potential function and the last step holds since $f(v, s_j) + f(v, s_\ell)$ is at most the congestion of v, which is at most 1.)

For the second part of the lemma, it suffices to note that the potential of the deactivated sink is less than the potential of the flow before the deactivation step. ■

Proof of Theorem 3: We first observe that for any non-sink node CONFLT marks exactly one outgoing edge (during a node aggregation step). Thus, the set of marked edges form a confluent flow. We now prove that the congestion of this flow is at most $\lg k + 1$. The potential of the flow at the start of the algorithm is at most $2k$. Lemmas 4 through 6 show that the potential of the flow is never increased. Furthermore, by Lemma 6, the potential of any deactivated sink is at most $2k$. Therefore, at termination, the potential of any sink is at most $2k$. The congestion of any sink in the final confluent flow is at most $1 + \lg k$. ■

4.2 An improved upper bound

In this section, we present a refinement Imp-Conflt of the algorithm Conflt and show that Imp-Conflt achieves $1+\ln k$ congestion. Imp-Conflt differs from Conflt in the sink deactivation step only. The remaining steps, namely breaking sawtooth cycles and node aggregation, are all identical to those in Conflt.

We now present the sink deactivation step for Imp-Conflt. As in the analysis of Conflt, we maintain a potential for each sink. The potential of a sink with congestion x is given by $\phi(x) = e^x$. The potential of the system is defined to be the sum of the potentials of the active sinks.

3. **Parsimonious sink deactivation:** Find an induced subgraph $G_1 \subseteq G$ such that every edge in G_1 joins a frontier node to a sink, G_1 contains all the outgoing edges of its frontier nodes and all the incoming edges of its sink nodes, and every frontier node in G_1 is adjacent to at least two sinks. Let S_1 be the set of sinks in G_1.
   - **Balancing:** Redistribute the flow from frontier nodes in G_1 to S_1 so that $\sum_{s_i \in S_1} \phi(b_i)$ is minimized. Remove any edges with zero flow.
   - If any edges are removed in the **Balancing** step, then go to the top of the main loop of Imp-Conflt.
   - Find a sink $s \in S_1$ with minimum total in-flow. For any frontier node v adjacent to s, redirect the flow along any edge (v, s) to an arbitrary sink adjacent to v. Remove all edges adjacent to s, deactivate s, thus setting $S_1 \leftarrow S_1 - \{s\}$.
   - Repeat **Balancing** and then repeat the main loop of Imp-Conflt.

To complete the description of the parsimonious sink deactivation step, we need to specify (i) how the subgraph G_1 is determined and (ii) how the step **Balancing** is implemented. For a directed (multi)graph D, let the *undirected graph underlying D* denote the undirected graph with vertex set $V(D)$ and with an edge for each pair of vertices which are joined (in either order) by a directed edge of D. The construction of G_1 is given by the proof of the following lemma.

Lemma 7 *If none of the preconditions of Lemmas 4 and 5 hold, then G contains a subgraph G_1 meeting the preconditions of the parsimonious sink deactivation step. Moreover, the undirected graph underlying G_1 is a tree.*

Proof: Recall the graph $\hat{G}$ defined in step 2a of Algorithm Conflt. We construct G_1 from $\hat{G}$ by the following linear-time algorithm, which is also illustrated in Figure 5.

1. Determine the graph H formed by the strongly connected components (SCCs) of $\hat{G}$: each vertex x of H corresponds to a strongly connected component C_x of $\hat{G}$ and there is an edge from a vertex x to a vertex y in H iff there is an edge in $\hat{G}$ from a vertex in SCC C_x to a vertex in SCC C_y.

2. By definition, H is acyclic. Find a node s of H with no outgoing edges. Set $\hat{G}_1$ to be the subgraph of $\hat{G}$ induced by C_s, and set $G_1 = \hat{G}_1 \cap G$.

Before we prove the desired claim of the lemma, we establish some properties of the graph H constructed in the first step of the above algorithm. Since the precondition of Lemma 5 does not hold, $\hat{G}$ does not have any directed cycle of length greater than 2. Since all length-2 cycles in $\hat{G}$ consist of a frontier node and a sink, H can be obtained from $\hat{G}$ by simply contracting all edges from frontier nodes to sinks and vice-versa, eliminating self-loops and retaining all other edges. Furthermore, for a given vertex x of H, if C_x contains more than one node then every node in C_x is either a frontier node or a sink. This is because there is no cycle in $\hat{G}$ that traverses a vertex that is neither a frontier node nor a sink.

We now show that G_1 satisfies the conditions of the parsimonious sink deactivation step. First, G_1 is an induced subgraph of G, by construction. We next note that every node in G_1 is either a frontier node or a sink; otherwise, C_s is a singleton set consisting of a node in $\hat{G}$ that is neither a sink nor a frontier node, but any such node has at least one outgoing edge, leading to a contradiction.

Next, we claim that every edge of G_1 joins a frontier node to a sink. If not, G_1 contains an edge $e = (v, w)$ between two frontier nodes because sink nodes have no outgoing edges. Since v and w map to the same vertex of H, they must be joined in $\hat{G}$ by a directed path from w to v. This path cannot be the edge (w, v) since every simple cycle of length 2 in $\hat{G}$ consists of a frontier node and a sink. Since this directed path from w to v is of length greater than one, appending the edge e to this path yields a directed simple cycle in $\hat{G}$ of length greater than 2, violating the hypothesis that the precondition of Lemma 5 is not met.

Next, we claim that G_1 contains all the outgoing edges of its frontier nodes; if it failed to contain such an edge e, then e would be an outgoing edge from s in H, violating the construction. It is also clear that G_1 contains all the incoming edges of its sinks, because these edges are all contracted in forming H. Finally, every frontier node in G_1 is adjacent to at least two sinks because otherwise the precondition of Lemma 4 would hold.

We complete the proof of the lemma by noting that the undirected graph underlying G_1 contains no cycles, because such cycles would correspond to cycles in $\hat{G}$ violating the precondition of Lemma 5. ■

For **Balancing**, we note that the minimization problem in the step is to minimize a convex cost function subject to certain linear constraints. We now show in Section 4.2.1 that the minimum of $\sum_{s_i \in S_1} e^{b_i}$ can be obtained by a polynomial time algorithm based on max flow. In Section 4.2.2, we show that the parsimonious sink deactivation step does not increase the potential of the resulting flow.

4.2.1 Convex minimization

In this section we consider the convex program that captures the balancing operation. We present a strongly polynomial-time algorithm for computing this minimum. We first characterize the structure of the minimizing point and show that it is "balanced" in a sense to be made precise in Definition 8 below. We then demonstrate that any balanced flow is a local optimum and hence also a global optimum. We conclude by showing that a balanced flow can be found in polynomial time using Megiddo's lexicographically optimal maximum flow algorithm [20]. An interesting consequence of our argument is that, in fact, the point of minimization is independent of the cost function.

Consider the bipartite graph formed by the frontier nodes $\{r_i\}$ and the sinks $\{s_j\}$. Let E' denote the set of pairs (i, j) such that there is an edge (r_i, s_j) in the bipartite graph. Let x_{ij} represent the flow on the edge from frontier node r_i to sink s_j. Following previous notation, let b_j represent the congestion of sink s_j and $d(s_j)$ denote the demand at sink s_j. Let c_i be the congestion at frontier node r_i. Let C be a nonnegative, increasing and strictly convex function; in particular, C could be the potential function ϕ defined earlier in Section 4.1 and 4.2. Consider the following program $\mathcal{P}^{\geq}$:

$$\begin{aligned} & \min \sum_j C(b_j) \\ \text{s.t.} & \\ \forall i, \quad & \sum_{(i,j)\in E'} x_{ij} \geq c_i \\ \forall j, \quad & b_j \geq \sum_{(i,j)\in E'} x_{ij} + d(s_j) \\ \forall (i,j) \in E', \quad & x_{ij} \geq 0 \end{aligned}$$

The above is a (strictly) convex program since a sum of (strictly) convex functions is (strictly) convex. It is well known that a strictly convex function over a convex set has a unique local minimum which is also the global minimum (e.g., see [1, Appendix B]). We also refer to this convex objective function as the potential function.

Note that without loss of generality we may replace the inequalities in the constraints above by equalities, yielding the following equivalent program $\mathcal{P}^{=}$:

$$\begin{aligned} & \min \sum_j C(b_j) \\ \text{s.t.} & \\ \forall i, \quad & \sum_{(i,j)\in E'} x_{ij} = c_i \\ \forall j, \quad & b_j = \sum_{(i,j)\in E'} x_{ij} + d(s_j) \\ \forall (i,j) \in E', \quad & x_{ij} \geq 0 \end{aligned}$$

Definition 8 *Consider a frontier node r_i that has edges with nonzero flow to two sinks s_j and s_k such that $b_j > b_k$; then we define the* balance *operation as the act of shifting flow from edge (i,j) to edge (i,k), until $x_{ij} = x_{ik}$ or feasibility along one of the edges is violated. We also refer to the operation as* balancing r_i. *We say that a flow is* balanced *if it cannot be balanced any further.*

Observe that the act of balancing reduces the value of the potential function since C is convex and increasing. The following "structure" lemma characterizes the form of balanced flows.

Lemma 9 *In any balanced flow, for any frontier node r_i, if $x_{ij} > 0$ and $x_{ik} > 0$, then $b_j = b_k$. That is, all the sinks that receive flow from a given frontier node have the same congestion. Furthermore, in any balanced flow, if $x_{ij} > 0$ then there does not exists any edge (i,k) such that $b_k < b_j$.*

Proof: Consider a frontier node. It cannot send flow to two sinks with differing congestion values; otherwise it could be balanced. Similarly, if r_i is sending flow to sink s_j then r_i cannot have an edge to a sink s_k with $b_k < b_j$ for otherwise it could be balanced. ■

The above structure lemma implies that we may define natural partitions of the frontier nodes and of the sinks, as follows. For any congestion value ℓ, let S_ℓ denote the set of sinks that have congestion ℓ, and let F_ℓ denote the set of frontier nodes that send flow to any sink in S_ℓ. By Lemma 9, $\{F_\ell\}$ forms a partition of all the frontier nodes. We now use Lemma 9 to show that a balanced flow cannot be improved by local perturbations.

Lemma 10 *A balanced flow is a local minimum.*

Proof: Let $\{x_{ij}\}$ be a balanced flow, and let $\{b_j\}$ denote the congestions at the sinks. We now show that for any perturbation to $\{x_{ij}\}$ (and correspondingly to $\{b_j\}$) the potential function is not reduced. Let μ be the number of frontier nodes, and choose $\varepsilon > 0$ small enough that the difference between any two distinct elements of the set $\{0\} \cup \{b : b = b_j \text{ for some sink } s_j\}$ is greater than $2\mu\varepsilon$. Let X denote the set of flows $\{x'_{ij}\}$ that satisfy $|x'_{ij} - x_{ij}| \le \varepsilon$ for all i and j. Let $\{x^*_{ij}\}$ denote the flow in X with minimum potential, and let $\{b^*_j\}$ denote the associated congestions at the sinks.

Consider two congestion values b_j and b_k such that $b_j < b_k$. It follows from our choice of ε that $b^*_j < b^*_k$ because $b^*_j \le b_j + \mu\varepsilon < b_k - \mu\varepsilon \le b^*_k$. By Lemma 9, there is no flow in $\{x_{ij}\}$ along any edge from a frontier node in F_{b_j} to any sink in S_{b_k}. We now argue by contradiction that $\{x^*_{ij}\}$ also does not carry any flow along any edge from a frontier node in F_{b_j} to a sink in S_{b_k}. Suppose there is an edge from a frontier node r_i in F_{b_j} to a sink node s_k in S_{b_k} such that $x^*_{ik} > 0$. Then there exists a sink node s_j in S_{b_j} such that $x_{ij} > x^*_{ij}$. If we increment x^*_{ij} by $\min\{\varepsilon, (b^*_k - b^*_j)/2, x^*_{ik}\}$ and decrement x^*_{ik} by the same amount, then we obtain a new flow in X that has smaller potential than $\{x^*_{ij}\}$, yielding a contradiction. We also obtain from Lemma 9 that the bipartite graph contains no edge from a frontier node in F_{b_j} to any sink in S_ℓ for any $\ell < b_j$. Therefore, $\sum_{s_j \in S_\ell} b_j = \sum_{s_j \in S_\ell} b^*_j$. Since b_j is identical for every sink s_j in S_ℓ, the convexity of C implies that the potential of $\{x_{ij}\}$ is at most that of $\{x^*_{ij}\}$, thus completing the proof. ■

It now follows that

Corollary 11 *A balanced configuration is the unique global minimum.*

Since the balanced property is independent of the specific convex function C, it also follows that

Corollary 12 *The point of minimization is independent of C.*

We now present an alternative characterization of this unique balanced configuration that is useful from an algorithmic standpoint.

Definition 13 *Consider a flow f on a network with sinks $s_1, s_2, ..., s_k$ and corresponding congestions $b_1, b_2, ..., b_k$. Let σ^f denote the k-tuple of numbers obtained by arranging $b_1, b_2, ..., b_k$ in increasing order. Then, f is said to be lexicographically optimal (or max-min fair) if σ^f is maximal in the lexicographical order.*

Megiddo [20] proved the following theorem (actually, a generalization with multiple sources and sinks, but we state only the single-source multiple-sink version here, since that is adequate for our needs):

Theorem 14 *Given a capacitated network flow problem with multiple sinks, a flow can be found in $O(n^5)$ time that is both lexicographically optimal and a maximum flow.*

We now present a transformation of our convex program $\mathcal{P}^=$ into a capacitated maximum flow problem $\mathcal{F}$: we create a super source which we connect to each frontier node r_i with capacity c_i and to each sink node s_j with capacity $d(s_j)$; we retain the edges, E', with infinite capacity; nodes s_j continue to be sink nodes.

Lemma 15 *A lexicographically optimal maximum flow of $\mathcal{F}$ is a feasible balanced configuration of $\mathcal{P}^=$.*

Proof: It is easy to see that any maximum flow of $\mathcal{F}$ is a feasible solution to $\mathcal{P}^{=}$. Furthermore, any lexicographically optimal flow corresponds to a balanced configuration since the balancing operation corresponds to improving the flow in the lexicographic order. ■

Putting Megiddo's theorem together with the above lemma it now follows that

Corollary 16 *$\mathcal{P}^{\geq}$ can be solved in (strongly) polynomial time.*

We have thus shown that the parsimonious sink dectivation step can be implemented in strongly polynomial time, implying that IMP-CONFLT is a strongly polynomial-time algorithm.

4.2.2 Nonpositive net change in the potential

In the following, we prove that the net change in potential as a result of the parsimonious sink deactivation step is nonpositive. Let G_1 be as defined at the start of the parsimonious sink deactivation step and let G_1' be the subgraph of G_1 consisting of all edges with positive flow after applying the balancing step to G_1. Let K denote any weakly connected component of G_1'. By Lemma 7, the undirected graph underlying G_1 is a tree. Since the balancing step can only remove edges from G_1, the undirected graph underlying K is also a tree. It also follows from the analysis of the balancing step that every sink in K has the same congestion. Let b denote this congestion. For any edge (v, s) in K from a frontier node v to a sink s, we define the *subtree rooted at* (v, s), $T_{(v,s)}$, as the tree consisting of all nodes that can be reached from v via undirected paths passing through the edge (v, s).

Lemma 17 *Let v denote an arbitrary frontier node and let s denote an adjacent sink in K. Let β denote the smallest in-flow*[3] *in K and $\alpha \leq \beta$ be a real number such that the flow on edge (v, s) is at most $1 - \alpha$. Then, we can inject an additional flow of α on edge (v, s) and assign the resulting additional demand to sinks in $T_{(v,s)}$ such that the additive increase in potential is at most $(\alpha/\delta)(\phi(b + \delta) - \phi(b))$, for any $\delta \in [1 - \beta, \alpha]$.*

Proof: The proof is by induction on the number of sinks in K. The assumptions about K require that there be at least two sinks in K. So, for the induction base, we consider the case when K has exactly two sinks. Since the sum of the in-flows of the two sinks is at most 1, it follows that $\beta \leq 1/2$. Since $\alpha \leq \beta$, $[1 - \beta, \alpha]$ is nonempty exactly when $\alpha = \beta = 1/2$. We can inject an additional flow of 1/2 into either of the sinks and increase the potential by at most $\phi(b+1/2) - \phi(b)$, which completes the desired claim.

We now consider the induction step. If $[1 - \beta, \alpha]$ is empty, then there is nothing to prove. Otherwise, let δ be any real in $[1 - \beta, \alpha]$. We are given a frontier node v and an adjacent sink s such that the flow along (v, s) is at most $1 - \alpha$. Let v_1 through v_j denote j frontier nodes other than v that are adjacent to s. For $1 \leq \ell \leq j$, we select $\alpha_\ell \leq f(v_\ell, s)$ such that $\sum_\ell \alpha_\ell = \alpha - \delta$. Such a selection is well-defined since the sum of the flows along these edges is at least $\beta - (1 - \alpha)$, which is at least $\alpha - \delta$ since $\delta \geq 1 - \beta$.

We increase flow along edge (v, s) by α and decrement the flow along the edges (v_ℓ, s) by α_ℓ. This increases the congestion of sink s by δ. The decrease in the flow along the edges (v_ℓ, s) implies that an additional flow of α_ℓ needs to be redirected to other sinks. For each frontier node v_ℓ, we select an arbitrary adjacent sink $s_\ell \neq s$ and inject an additional flow of α_ℓ along edge (v_ℓ, s_ℓ) into the subtree rooted at edge (v_ℓ, s_ℓ) (i.e., subtree $T_{(v_\ell,s_\ell)}$). Let β_ℓ denote the smallest in-flow in the subtree rooted at edge (v_ℓ, s_ℓ). Since $\alpha_\ell \leq \alpha$ and $\alpha \leq \beta \leq \beta_\ell$, it follows that $\alpha_\ell \leq \beta_\ell$. By the

[3] Recall that the in-flow of a node v is the sum of the flows on the edges into v.

induction hypothesis, we know that the total increase in potential of the sinks in this subtree is at most

$$\frac{\alpha_\ell}{\delta'}\left(\phi(b+\delta')-\phi(b)\right),$$

for any $\delta' \in [1-\beta_\ell, \alpha_\ell]$. Since $\beta_\ell \geq \beta$ and $\delta \geq 1-\beta$, we obtain that $\delta \geq 1-\beta_\ell$. We consider two cases. If $\delta \leq \alpha_\ell$, it follows from the induction hypothesis that the total increase in potential in the subtree is at most

$$\frac{\alpha_\ell}{\delta}\left(\phi(b+\delta)-\phi(b)\right).$$

If $\delta > \alpha_\ell$, then we can simply increase the in-flow into sink s_ℓ by α_ℓ and achieve a potential increase of at most

$$\phi(b+\alpha_\ell)-\phi(b) \leq \frac{\alpha_\ell}{\delta}\left(\phi(b+\delta)-\phi(b)\right),$$

the last inequality following from Lemma 19 since $\delta > \alpha_\ell$.

By adding over all ℓ, we obtain that the total potential increase over all trees $T_{(v_\ell,s_\ell)}$ is at most

$$\frac{\sum_\ell \alpha_\ell}{\delta}\left(\phi(b+\delta)-\phi(b)\right).$$

On adding the potential increase of sink s, which is $\phi(b+\delta)-\phi(b)$, we obtain a total increase of at most

$$\frac{\alpha}{\delta}\left(\phi(b+\delta)-\phi(b)\right),$$

which completes the induction step. ■

Lemma 18 *The parsimonious sink deactivation step does not increase the potential.*

Proof: If any edges are removed after the first balancing operation of the parsimonious deactivation step, then the step terminates without increasing the potential. For the remainder of the proof, we assume that no edges are removed in the first balancing operation. Let s^* denote a sink in K with minimum in-flow. We now show that the net potential change following the deactivation of s^* and the redistribution of flow using the second balancing operation is nonpositive. Since the balancing operation redistributes flow so as to minimize the potential of the sinks, it suffices to show that there exists a flow redistribution following the deactivation of s^* that has the desired effect. Let α denote the in-flow of s^* and let v_1 through v_j denote the frontier nodes adjacent to s^*. Let α_ℓ denote the flow on edge (v_ℓ, s^*), for $1 \leq \ell \leq j$. Let $s_\ell \neq s^*$ denote an arbitrary sink adjacent to v_ℓ. (Since every frontier node in K has at least two adjacent nodes and no edges are removed in the first balancing operation, s_ℓ exists.) We redirect flow α_ℓ into the subtree $T_{(v_\ell,s_\ell)}$. Since the congestion of v_ℓ is at most 1, it follows that the flow along edge (v_ℓ, s_ℓ) is at most $1-\alpha_\ell$. We consider two cases. If $\alpha_\ell < 1-\alpha$, then we assign α_ℓ to sink s_ℓ, thus increasing its congestion to $b+\alpha_\ell$. Thus, the increase in potential of $T_{(v_\ell,s_\ell)}$ is at most

$$\phi(b+\alpha_\ell)-\phi(b) \leq \frac{\alpha_\ell}{1-\alpha}\left(\phi(b+1-\alpha)-\phi(b)\right),$$

by Lemma 19.

We now consider the case $\alpha_\ell \geq 1-\alpha$. Let β_ℓ denote the minimum in-flow of a sink node in $T_{(v_\ell,s_\ell)}$. Since $\beta_\ell \geq \alpha$, it follows that $1-\alpha \geq 1-\beta_\ell$. We invoke Lemma 17, with $\delta = (1-\alpha)$ to obtain that the increase in potential of $T_{(v_\ell,s_\ell)}$ is at most

$$\frac{\alpha_\ell}{1-\alpha}\left(\phi(b+1-\alpha)-\phi(b)\right).$$

Adding over all ℓ, and subtracting the potential of the deactivated sink s^*, we obtain the net change in the total potential of the system to be at most

$$\begin{aligned}
& \frac{\alpha}{1-\alpha}\left(\phi(b+1-\alpha)-\phi(b)\right)-\phi(b) \\
= \; & e^b\left(\frac{\alpha}{1-\alpha}(e^{1-\alpha}-1)-1\right) \\
= \; & e^b\left(\frac{\alpha}{1-\alpha}(\frac{1}{e^{\alpha-1}}-1)-1\right) \\
\leq \; & 0.
\end{aligned}$$

(The last step uses the inequality $e^{\alpha-1} \geq 1+\alpha-1 = \alpha$.) ■

Lemma 19 *For $0 \leq x \leq 1$, the function $\frac{e^x-1}{x}$ is a monotonically increasing function of x.*

Proof: Let $g(x)$ equal $\frac{e^x-1}{x}$. Then $g'(x)$ is equal to

$$\frac{e^x}{x}-\frac{e^x}{x^2}+\frac{1}{x^2}=\frac{e^x(x-1)+1}{x^2}=\frac{(x-1)/e^{-x}+1}{x^2},$$

which is nonnegative since $e^{-x} \geq 1-x$ and $x \leq 1$. ■

Theorem 20 *Given a splittable flow with node congestion 1 on a graph with k sinks, there exists a strongly polynomial time algorithm that finds a confluent flow satisfying the same demand and with node congestion not exceeding $1+\ln k$.*

Proof: Lemmas 4, 5, and 18 show that the potential never increases. Since the initial total potential is ek, it follows that potential of any sink at termination is at most ek, implying a congestion of at most $1+\ln k$. ■

5 Demand maximization

In the demand maximization problem, we seek a confluent flow which maximizes the total demand of the satisfied nodes, among all confluent flows which satisfy a subset of the nodes and meet some prescribed upper bound on the congestion. We assume without loss of generality that the prescribed upper bound on congestion is 1; otherwise, we can scale all demands and flows by the congestion bound. Suppose we are given a graph G that admits a splittable flow satisying all demands, while incurring a congestion of at most 1. We present a polynomial time algorithm for finding a confluent flow in G of congestion at most 1 that satisfies demands of a subset of nodes whose demands add up to (1/3)-factor of the total demand. We note that the preceding result assumes the existence of a splittable flow that satisfies all demands with congestion at most 1. As mentioned in Section 1, we can eliminate this assumption by first determining a splittable flow of congestion at most 1 that satisfies a constant fraction of the maximum satisfiable demand. For this purpose, we can use any of the constant-factor approximations known for demand maximization in single-source unsplittable flows [3, 15]; the best approximation ratio known is 4.43 [3]. This yields a 13.29-approximation for unconditional demand maximization in confluent flows.

The demand maximization algorithm MaxDemand is the same as Conflt except the sink deactivation step and a postprocessing step.

3. **Sink deactivation:** Let s_j be a sink node with only one adjacent frontier node v, and let s_ℓ be another sink node adjacent to v.

(a) If $b_j - f(v, s_j) \leq \frac{1}{2}$, remove edge (v, s_ℓ) and send its flow along edge (v, s_j).

(b) If $b_j - f(v, s_j) > \frac{1}{2}$, remove edge (v, s_j), send its flow along edge (v, s_ℓ) and deactivate sink s_j.

After obtaining a set of disjoint arborescences $\{T_i\}_{i=1}^k$, the algorithm MaxDemand selects a subset of the nodes and satisfies their demands as follows. Let $\hat{b}_i$ denote the total demand in T_i (alternatively, the congestion of sink s_i) after the computation of the arborescences. For each T_i, if $\hat{b}_i \leq 1$, then satisfy all of the demand. If $1 < \hat{b}_i \leq \frac{3}{2}$, then greedily partition the nodes in T_i into groups whose total demands sum to at most $\frac{2}{3}\hat{b}_i$; select the nodes in the group with the maximum total demand and satisfy their demands using T_i. If $\hat{b}_i > \frac{3}{2}$, then search the nodes of T_i in any order and determine a subset of the nodes whose total demand is at least 1/2 as follows.

```
1. $\Delta_i \leftarrow 0$.
2. While $\Delta_i < 1/2$ do
   (a) Visit the next node $v$.
   (b) If $d(v) \leq 1/2$, then mark $v$ and $\Delta_i \leftarrow \Delta_i + d(v)$. Otherwise, unmark all marked nodes;
       mark $v$ and $\Delta_i \leftarrow d(v)$.
3. Select the marked nodes and satisfy their demands using $T_i$.
```

Lemma 21 *After the calculation of the arborescences, we have*

$$\sum_{i:\hat{b}_i \leq 1} \hat{b}_i + \frac{1}{2} \sum_{i:1<\hat{b}_i\leq\frac{3}{2}} \hat{b}_i + \sum_{i:\hat{b}_i>\frac{3}{2}} \frac{1}{2} \geq \frac{D}{3}.$$

Proof: We first show:

$$2 \sum_{i:\frac{1}{2}<\hat{b}_i\leq 1} \hat{b}_i + \frac{1}{2} \sum_{i:1<\hat{b}_i\leq\frac{3}{2}} \hat{b}_i \geq \sum_{i:\hat{b}_i>\frac{3}{2}} (\hat{b}_i - \frac{3}{2})$$

We plot the congestion of the sinks $\{b_i\}_{i=1}^k$ as a histogram. These values may change during the execution of the algorithm MaxDemand, specifically, in step 3. But the total area of the histogram is always D. Divide the xy-plane of the histogram into two sections by the $y = \frac{3}{2}$ line. Notice that when the algorithm redirects flow from one sink to another, demand moves from one sink to the other; in the histogram, this corresponds to a move of "area" either across the boundary or within a section. Denote by A the total area of the upper section. The area A cannot increase in step 3a since $b_j + f(v, s_\ell) \leq \frac{1}{2} + f(v, s_j) + f(v, s_\ell) \leq \frac{3}{2}$. Whenever A increases in step 3b, some sink s_j is deactivated and remains untouched thereafter, hence $\hat{b}_j = b_j - f(v, s_j)$. There are now three cases to consider.

1. If $\frac{1}{2} < b_j - f(v, s_j) \leq 1$, then the increase in A is at most $1 \leq 2\hat{b}_j$.

2. If $1 < b_j - f(v, s_j) \leq \frac{3}{2}$, then the increase in A is at most $f(v, s_j) - (b_j - \frac{3}{2}) < \frac{1}{2} < \frac{1}{2}\hat{b}_j$.

3. If $b_j - f(v, s_j) > \frac{3}{2}$, then A does not increase.

Putting these together we obtain

$$2 \sum_{i:\frac{1}{2}<\hat{b}_i\leq 1} \hat{b}_i + \frac{1}{2} \sum_{i:1<\hat{b}_i\leq\frac{3}{2}} \hat{b}_i \geq \sum_{i:\hat{b}_i>\frac{3}{2}} (\hat{b}_i - \frac{3}{2})$$

Note that

$$\sum_{i:\frac{1}{2}<\hat{b}_i\leq 1} \hat{b}_i + \sum_{i:1<\hat{b}_i\leq\frac{3}{2}} \hat{b}_i + \sum_{i:\hat{b}_i>\frac{3}{2}} \hat{b}_i = D$$

Adding the two equations and dividing both sides by 3 yields the desired result. ■

Theorem 22 *Given a splittable flow with node congestion 1 on a graph with total demand D,* MaxDemand *finds a confluent flow with node congestion at most 1, satisfying demands of a subset of nodes, with total demand at least $\frac{D}{3}$.*

Proof: Since the demands are routed with $\{T_i\}_{i=1}^k$, the flow is confluent. Since for each T_i, the algorithm selects total demand at most 1, the node congestion is at most 1. For T_i with $\hat{b}_i \leq 1$, all $\hat{b}_i$ demand is satisfied. For T_i with $1 < \hat{b}_i \leq \frac{3}{2}$, suppose the total demands of the groups are $d_1 \leq d_2 \leq \cdots \leq d_r$. Only d_1 can be less than or equal to $\hat{b}_i/3$, otherwise d_1 and d_2 could have been combined. Since $d_1 + d_r > \frac{2}{3}\hat{b}_i$, it follows that $r = 2$ (otherwise $d_1 + d_2 + d_r > \hat{b}_i$). Therefore, $d_r \geq \hat{b}_i/2$, i.e. at least $\hat{b}_i/2$ is satisfied; totally $\frac{1}{2}\sum_{i:1<\hat{b}_i\leq\frac{3}{2}} \hat{b}_i$ is satisfied. For T_i with $\hat{b}_i > \frac{3}{2}$, at least $\Delta_i \geq 1/2$ demand is satisfied; totally at least $\sum_{i:\hat{b}_i>\frac{3}{2}} 1/2$ is satisfied. From Lemma 21, the algorithm satisfies at least $D/3$ demand in total. ■

6 Confluent flows in k-connected graphs

It turns out that if G is k-connected, then any splittable flow of congestion C may be replaced by a confluent flow of congestion less than $C + d_{\max}$. In fact we will prove a considerably stronger theorem. Let us say that a directed graph G is *k-connected to a set $S \subseteq V(G)$*, if each vertex $v \notin S$ can be joined to S by k paths which are disjoint except for the common vertex v. Throughout this section, we will use the notation $C_f(v)$ to denote the congestion induced by a flow f at a vertex v; the subscript f will be omitted when it is clear from context.

Theorem 23 *Let G be a directed graph with sinks $S = \{s_1, \dots, s_k\}$, demands $d : V \to \mathbb{R}_+$, maximum demand $d_{\max}$, and total demand D. Suppose that G is k-connected to S. Given target values $t_1, \dots, t_k$ summing to D, there is a confluent flow in G such that $C(s_i) < t_i + d_{\max}$ for all i.*

In particular, taking $t_1 = \ldots = t_k = D/k \leq C$, we obtain a confluent flow of congestion less than $C + d_{\max}$ as claimed. Theorem 23 can be seen as a weighted version of the following theorem.

Theorem 24 (Lovász, 1977 [19]) *Let G be a digraph, $S = \{v_1, \dots, v_k\} \subseteq V(G)$ a set of k vertices, and assume that G is k-connected to S. Let, furthermore, k positive integers $n_1, \dots, n_k$ be given whose sum is $|V(G)|$. Then G contains k vertex-disjoint arborescences $A_1, \dots, A_k$, such that A_i is rooted at v_i and $|V(A_i)| = n_i$.*

Before embarking on the proof of Theorem 23, which is somewhat elaborate and involves a non-constructive step, we pause here to introduce the main ideas of the proof and to place them in the context of the foregoing sections of this paper. Let G be a directed graph and let f be a splittable flow in G with congestion C. Let us call a vertex v *undecided* if the flow f assigns a positive flow value to more than one outgoing edge of v. Looking closely at the iterative rounding procedures used in Sections 4.1 and 4.2, one sees that the problematic cases for those algorithms are cases in which there is a long flow path P containing many undecided vertices, and each of them decides, during the execution of the algorithm, to send all of its flow on the edges of P. In such cases, the congestion of the final vertex of P increases with each such rounding decision, eventually becoming as large as $\Omega(C \log k)$ in the worst case. Indeed, Section 3.1 presents an example in which this logarithmic increase in congestion is unavoidable. However, in case the original flow f contains no path P with many undecided nodes, we can hope to achieve better congestion when rounding f to a confluent flow. Let us call a flow in G *nearly confluent* if every undecided node has zero incoming flow. (Note that this propery implies that every flow path P contains at most one undecided node, namely the first node of the path.) We will prove in Theorem 26 below that every such flow can be rounded to a confluent flow while increasing the congestion of every sink by at most $d_{\max}$. The proof uses a simple rounding technique in the spirit of [17]. To complete the proof of Theorem 23, we then establish the following existence theorem for nearly confluent flows in k-connected graphs.

Theorem 25 *Let G be a directed graph with sinks $S = \{s_1, \ldots, s_k\}$, demands $d : V \to \mathbb{R}_+$ and total demand D. Suppose that G is k-connected to S. Given any non-negative target values $t_1, \ldots, t_k$ summing to D, there is a nearly confluent flow f in G such that $C(s_i) = t_i$ for all i.*

The proof of Theorem 25 is non-constructive, making use of techniques originally introduced by Lovász in proving Theorem 24. The idea of the proof is topological: the set of nearly confluent flows constitutes a topological space $\mathcal{K}$, the function which maps a nearly confluent flow to the vector of congestions $(C(s_1), \ldots, C(s_k))$ is a continuous function from $\mathcal{K}$ to $\mathbb{R}^k$, and we aim to prove that there exists a point of $\mathcal{K}$ which maps to a designated point $(t_1, t_2, \ldots, t_k) \in \mathbb{R}^k$. For a continuous function from $\mathcal{K}$ to $\mathbb{R}$, we could hope to establish the existence of such a point using the intermediate value theorem, provided that $\mathcal{K}$ is path-connected (i.e., any two points of $\mathcal{K}$ can be joined using a continuous path). For our purposes, we need to use a multi-dimensional version of the intermediate value theorem (see Theorem 29) and we need the space $\mathcal{K}$ to satisfy higher-dimensional connectivity properties in addition to path-connectivity. These higher-dimensional connectivity properties are expressed using homology theory [13], and for k-connected graphs they were established by Lovász in the course of proving Theorem 24.

6.1 Rounding nearly confluent flows

Theorem 26 *If f is a nearly confluent flow in G, then f may be rounded to a confluent flow $\tilde{f}$ such that $C_{\tilde{f}}(s_i) < C_f(s_i) + d_{\max}$ for all i.*

Proof: We describe an algorithm for rounding f to $\tilde{f}$. In describing this algorithm it will be useful to adjoin to G a vertex $\hat{a}$ with one incoming edge from each vertex in the set of sinks $S = \{s_1, \ldots, s_k\}$. The flow f is extended to this augmented graph by specifying that the flow on the edge from s_i to $\hat{a}$ is $C_f(s_i)$.

Delete all edges e with $f(e) = 0$ from G. Let $U \subseteq V(G)$ denote the set of nodes with outdegree greater than 1; as above, we will call these the *undecided nodes.* The algorithm will select undecided nodes one by one, and will re-route all the flow from the selected node along one of its outgoing edges. It follows that the flow value will never increase on an edge which is not downstream from an undecided node; call such edges *safe*, and the other edges of G *active.* (Here, we say that a node v is *downstream* of a node u if there is a directed path in G from u to v; we also express this relation by saying u is *upstream* from v.)

Throughout this proof, we will use the term *weak cycle* to refer to a sequence of vertices $v_0, v_1, \ldots, v_{n-1}, v_n = v_0$ in $V(G)$ such that for $i = 1, 2, \ldots, n$, either (v_{i-1}, v_i) or (v_i, v_{i-1}) is a directed edge of G. (In the former case, we call (v_{i-1}, v_i) a *forward edge* of the weak cycle; in the latter case we call it a *reverse edge.*) When G contains a weak cycle C with a positive flow value on each edge, we can *augment* the flow along this cycle by adding some positive amount $\varepsilon > 0$ to the flow value on each forward edge and subtracting the same amount ε from each reverse edge. As long as G contains a weak cycle which does not pass through $\hat{a}$, we can modify f by finding an edge on this cycle with minimum flow value f_{min}, performing an augmentation along the cycle with $\varepsilon = f_{min}$, and deleting all edges whose flow value is now zero. Each such operation diminishes $|E|$ by at least one, and doesn't alter the congestion of any vertex in S. (The congestion of s_i is equal to the flow value on the edge from s_i to $\hat{a}$, and this edge is not in the augmenting cycle, so the augmentation doesn't change its flow value.) After finitely many such operations it will be the case that every weak cycle in G passes through $\hat{a}$.

Now we claim there exists a vertex $s_i \in S$ with only one undecided vertex upstream from it. This can be proved by considering the undirected bipartite graph B with vertex set $U \cup S$ whose edge set consists of all pairs $\{u, s\}$ such that $u \in U, s \in S$, and G contains a directed path from u to s. This bipartite graph B must be a forest: for any simple cycle C in B, the edges of C correspond to paths in G whose union contains a weak cycle that does not pass through $\hat{a}$, contradicting our earlier assumption. Moreover, every vertex $u \in U$ has at least two neighbors in B. This is because u has at least two outgoing edges in G, each of these edges is the beginning of a directed path leading to a sink, and these sinks must be distinct because otherwise the union of the two directed paths in G would contain a weak cycle which does not pass through $\hat{a}$, again contradicting our assumption. It follows that every leaf of the forest B is a sink $s_i \in S$. By the definition of B, this implies that s_i has only one undecided vertex upstream from it in G.

Having found a vertex $s_i \in S$ with only one undecided vertex $u \in U$ upstream from it, we take all of the demand at u, re-route it along the path from u through s_i to $\hat{a}$, delete u from the set of undecided vertices, and delete the edges of this path from the set of active edges. The algorithm continues in this manner until U is empty, at which time f has been rounded to a confluent flow. We then delete $\hat{a}$ and all of its incoming edges to obtain a confluent flow $\tilde{f}$ in the original graph G.

Note that if $s_i \in S$, then the congestion of s_i only increases during the step when the edge $e = (s_i, \hat{a})$ changes from an active edge into a safe edge. At this step, the increase in congestion is less than $d(u)$, the amount of demand at the undecided vertex which re-routed its flow along a path through e. Since the initial congestion of s_i is $C_f(s_i)$ and the final congestion of e is $C_{\tilde{f}}(s_i)$, we obtain $C_{\tilde{f}}(s_i) < C_f(s_i) + d_{\max}$, as desired. ■

6.2 The arborescence complex of a directed graph

The remaining steps in the proof of Theorem 23 rely heavily on methods from algebraic topology, especially homology theory. In the following proofs, we assume familiarity with the definition of homology and relative homology groups, as well as basic facts about homology groups such as the excision theorem and the long exact sequence of a pair. For an exposition of these topics, we refer the reader to Hatcher's textbook on algebraic topology [13].

In [19] a topological space $\mathcal{K}$ is associated with each directed graph G with distinguished vertex $\hat{a}$, known as the *arborescence complex* of G relative to $\hat{a}$. It can be modeled as a CW complex [4] whose vertices (the *0-cells* of $\mathcal{K}$, in the terminology of CW complexes) are in one-to-one correspondence with the arborescences of G rooted at $\hat{a}$. In this section we present a definition of the arborescence complex which is equivalent to the original, but makes clearer the relation with confluent flows.

[4]CW complexes are a generalization of simplicial complexes, in which the n-dimensional building blocks ("n-cells") of the space are homeomorphic to n-dimensional simplices, but their boundaries may be attached to the lower-dimensional cells using arbitrary continuous mappings, unlike in the case of a simplicial complex X, whose n-dimensional cells have boundaries made up of $(n-1)$-dimensional simplices each of which is identified (homeomorphically) with one of the $(n-1)$-dimensional simplices of X. For a precise definition of CW complexes, we refer the reader to Hatcher's book [13].

Definition 27 *Let $G = (V, E)$ be a directed acyclic graph with distinguished vertex $\hat{a}$. Assume every $v \in V(G)$ has a directed path to $\hat{a}$. Represent each $\hat{a}$-rooted arborescence by a function $F : E(G) \to \mathbb{R}$ where $F(e) = 1$ if e belongs to the arborescence, 0 otherwise. A* fractional arborescence *in G is a function $F : E(G) \to [0, 1]$ which is a convex combination of arborescences.*

A fractional arborescence is called a near-arborescence *if it satisfies the following property: if v is a vertex such that $F(e) > 0$ for at least two distinct outgoing edges e, then $F(e) = 0$ for all incoming edges e. In other words, a near-arborescence is a convex combination of arborescences, any of which can be transformed into any other by disconnecting and reattaching some leaves. The set of all near-arborescences in G is a subspace $\mathcal{K} \subseteq \mathbb{R}^{E(G)}$ called the* arborescence complex *of G relative to $\hat{a}$.*

Definition 27 constitutes a precise description of the arborescence complex $\mathcal{K}$ as a topological space. As an aid in visualization, there is a natural decomposition of $\mathcal{K}$ as a union of polyhedral pieces, which we now describe. (This decomposition also serves to specify the CW-complex structure of $\mathcal{K}$.) For a near-arborescence F, we may define the *support* of F to be the set of edges $e \in E$ such that $F(e) > 0$. If $E_0 \subseteq E$ is an edge set, let us consider the set $\mathcal{K}[E_0]$ of all near-arborescences with support E_0. This set is empty if there is a vertex with at least one incoming edge and two outgoing edges in E_0. Otherwise, $\mathcal{K}[E_0]$ is equal to the set of all ways of assigning positive real-valued weights to the edges in E_0 such that the sum of the weights of the edges exiting every vertex except $\hat{a}$ is exactly 1. In other words,

$$\mathcal{K}[E_0] = \prod_{v \in V(G) \setminus \hat{a}} \operatorname{int}(\Delta^{d_0(v)-1}),$$

where $d_0(v)$ denotes the number of outgoing edges of a vertex v and $\operatorname{int}(\Delta^k)$ denotes the standard open simplex

$$\operatorname{int}(\Delta^k) = \{(x_0, x_1, \ldots, x_k) \mid x_0 + \ldots + x_k = 1,\ x_0, x_1, \ldots, x_k > 0\}.$$

Thus $\mathcal{K}$ can be partitioned into subsets $\mathcal{K}[E_0]$ indexed by certain edge sets E_0, each such subset is the interior of a convex polytope of dimension $|E_0| - |V(G)| + 1$, and the boundary of $\mathcal{K}[E_0]$ is equal to the union of $\mathcal{K}[E']$ as E' runs through all the subsets of E_0. For future reference, when F is a near-arborescence in $\mathcal{K}[E_0]$ we will refer to a vertex v as *undecided* if $d_0(v) > 1$.

For example the vertices of the arborescence complex are the arborescences in G rooted at $\hat{a}$. Two such arborescences A_1, A_2 are joined by a line segment in $\mathcal{K}$ if there is a vertex v in G with two outgoing edges e, e', and an arborescence A in $G - \{v\}$, such that $A_1 = A \cup \{e\}$, $A_2 = A \cup \{e\}$. Similarly, $\mathcal{K}$ contains a triangle for every three arborescences obtained by taking a vertex v with three outgoing edges e, e', e'', and an arborescence A in $G - \{v\}$, and joining v to A using each of the three edges; and $\mathcal{K}$ contains a square for every four arborescences obtained by taking a pair of vertices v_1, v_2, each with two outgoing edges, and an arborescence A in $G - \{v_1, v_2\}$, and joining v_1, v_2 to A in each of the four possible combinations.

In Figure 6 we have illustrated the arborescence complex for a six-vertex graph G. In this case, a fractional arborescence is determined by specifying, for each of the three topmost vertices of G, a convex combination of the two outgoing edges. Thus the space of fractional arborescences in G is a cube: a product of three copies of Δ^1, one for each of the three topmost vertices. The near-arborescences are those in which the topmost vertex does not feed any weight into an undecided vertex. The arborescence complex $\mathcal{K}$ is the subset of the cube where this criterion holds. It is an octagon consisting of eight edges of the cube; these eight edges are denoted by solid lines in Figure 6.

The fundamental fact about the topology of arborescence complexes is the following theorem from [19].

Theorem 28 ([19], Theorem 4) *Let G be a digraph, $\hat{a} \in V(G)$, and assume that G is k-connected to $\hat{a}$ ($k \geq 2$). Then the arborescence complex $\mathcal{K}$ of G relative to $\hat{a}$ satisfies $\tilde{H}_0(\mathcal{K}) = \ldots = \tilde{H}_{k-2}(\mathcal{K}) = 0$, where $\tilde{H}$ denotes reduced homology with integer coefficients.*

Here, as in [19], the importance of Theorem 28 is that it enables us to apply a generalized intermediate value theorem to obtain a near-arborescence with desired properties.
Proof of Theorem 25: Given a graph G with demands $d : V(G) \to \mathbb{R}_+$ and sinks $s_1, \ldots, s_k$, extend it to a graph $\hat{G}$ by adjoining an auxiliary vertex $\hat{a}$ with incoming edges $e_1, \ldots, e_k$ from $s_1, \ldots, s_k$. In terms of $\hat{G}$, we want to show that there is a nearly confluent flow f in $\hat{G}$ satisfying $f(e_i) = t_i$ for $i = 1, \ldots, k$.

If one is given demands $d : V(\hat{G}) \to \mathbb{R}_+$ and a near-arborescence F, this data naturally defines a flow $f : E(\hat{G}) \to \mathbb{R}_+$, routing all the demands to the sink vertex $\hat{a}$, according to the prescription that each vertex v distributes its outgoing flow in the proportions specified by F. (Note that the set of edges e with $F(e) > 0$ must contain no directed cycles in order for this flow to be well-defined. If F is a near-arborescence, it is automatic that this edge set contains no directed cycles.) For a fixed set of demands, the mapping which associates to each near-arborescence F the corresponding flow f constitutes a continuous function from $\mathcal{K}$ to the space of all flows in G. If F is an arborescence, the corresponding flow f is a confluent flow. If F is a near-arborescence, the corresponding flow is nearly confluent.

Let $\Delta \subseteq \mathbb{R}^k$ denote the $(k-1)$-simplex

$$\Delta = \left\{ (x_1, \ldots, x_k) \in \mathbb{R}_+^k \, : \, \sum_{i=1}^{k} x_i = D \right\}.$$

If $\mathcal{K}$ is the arborescence complex of $\hat{G}$ relative to $\hat{a}$, we may map $\mathcal{K}$ to Δ by mapping a near-arborescence F to the vector $(f(e_1), f(e_2), \ldots, f(e_k))$, where f is the flow corresponding to F. This is a continuous function $\phi : \mathcal{K} \to \Delta$, and the theorem asserts that there is a point $F \in \mathcal{K}$ which maps to $(t_1, \ldots, t_k)$. The existence of such a point is proved using the following generalized intermediate value theorem, whose proof is given below.

Theorem 29 *Let X be a finite-dimensional cellular complex. Suppose $\phi : X \to \Delta^r$ is a continuous map to an r-dimensional simplex, such that for every s-dimensional face $\sigma^s \subseteq \Delta^r$, the subspace $Y = \phi^{-1}(\sigma^s) \subseteq X$ is a non-empty subcomplex satisfying $\tilde{H}_0(Y) = \ldots = \tilde{H}_{s-1}(Y) = 0$. Then ϕ is a surjection, i.e. every point of Δ^r is the image of some point in X under ϕ.*

To explain why we refer to this as a generalized intermediate value theorem, consider the case $r = 1, \Delta^r = [0, 1]$. Then the hypotheses of the theorem amount to:

- $f^{-1}(0)$ is non-empty.
- $f^{-1}(1)$ is non-empty.
- $\tilde{H}_0(X) = 0$, i.e. X is path-connected.

In other words, the theorem asserts that if a continuous function on a path-connected space takes the values 0 and 1, then it also takes every value between 0 and 1. This is standard intermediate value theorem.

To apply Theorem 29 to the arborescence complex $\mathcal{K}$ and the mapping $\phi : \mathcal{K} \to \Delta^{k-1}$ defined by $F \mapsto (f(e_1), \ldots, f(e_k))$, we must verify that $\phi^{-1}(\sigma^s)$ satisfies the homological vanishing criterion specified in the theorem, for each face $\sigma^s \subseteq \Delta^{k-1}$. Each such face is determined by specifying $s + 1$

of the incoming edges at $\hat{a}$ — without loss of generality, say $e_1, \ldots, e_{s+1}$ — and requiring the flow to be zero on e_i for all $i > s+1$. Thus $\phi^{-1}(\sigma^s)$ is the arborescence complex of $G \cup \{e_1, \ldots, e_{s+1}\}$ relative to $\hat{a}$. In this graph, no $v \neq \hat{a}$ can be separated from $\hat{a}$ by removing fewer than $s+1$ vertices, so Theorem 28 ensures the vanishing of the required homology groups. ■

Proof of Theorem 29: Suppose we are given a map $\phi : X \to \Delta^r$ satisfying the homological vanishing criterion of Theorem 29, i.e. for any face $\sigma^s \subseteq \Delta^r$, the inverse image $Y = \phi^{-1}(\sigma^s)$ satisfies $\tilde{H}_0(Y) = \ldots = \tilde{H}_{s-1}(Y)$. We will prove ϕ is surjective by induction on r. When $r = 0$ there is nothing to prove, as Δ^r consists of a single point. For $r > 0$, we may apply the induction hypothesis on each face of the boundary $\partial\Delta^r$ to conclude that the image of ϕ contains every point of $\partial\Delta^r$. So now assume that ϕ misses some point p in the interior of Δ^r; we'll derive a contradiction in a manner similar to the proof of Brouwer's fixed point theorem.

The starting point is a homological computation encapsulated in the following lemma.

Lemma 30 *The map $\phi_* : \tilde{H}_{r-1}(\phi^{-1}(\partial\Delta^r)) \to \tilde{H}_{r-1}(\partial\Delta^r) = \mathbb{Z}$ is a surjection.*

Proof: We will prove the following more general claim: if $A \subseteq \Delta^r$ is an s-dimensional subcomplex containing the $(s-1)$-skeleton of Δ^r (i.e. the union of all faces of dimension $< s$) then the map $\phi_* : \tilde{H}_j(\phi^{-1}(A)) \to \tilde{H}_j(A)$ is an isomorphism for $j < s$ and a surjection for $j = s$. The proof is by induction on s and on the number of s-dimensional cells of A. In the base case, $s = 0$, the functor $\tilde{H}_0$ associates to each space a free abelian group whose rank is one less than the number of connected components of that space; the claim now follows from the observation that $\phi^{-1}(v)$ is non-empty for each 0-simplex (i.e. vertex) $v \in \Delta^r$. For the induction step, suppose $s > 0$, let σ^s be any s-simplex of A, and let $B = A - \sigma^s$. We have the following commutative diagram whose top and bottom rows are exact sequences (the homology long exact sequences of the pairs $(\phi^{-1}(A), \phi^{-1}(B))$ and (A, B), respectively), and whose vertical arrows are homomorphisms induced by ϕ.

$$\begin{tikzcd}
\tilde{H}_j(\phi^{-1}(B)) \arrow[r] \arrow[d] & \tilde{H}_j(\phi^{-1}(A)) \arrow[r] \arrow[d] & \tilde{H}_j(\phi^{-1}(A), \phi^{-1}(B)) \arrow[r, "\partial"] \arrow[d] & \tilde{H}_{j-1}(\phi^{-1}(B)) \arrow[d, "\approx"] \\
\tilde{H}_j(B) \arrow[r] & \tilde{H}_j(A) \arrow[r] & \tilde{H}_j(A, B) \arrow[r, "\partial"] & \tilde{H}_{j-1}(B)
\end{tikzcd}$$

By the induction hypothesis, the vertical map on the right is an isomorphism, and the one on the left is an isomorphism for $j < s$ and a surjection for $j = s$. If we can prove that the third vertical map, $\phi_* : \tilde{H}_j(\phi^{-1}(A), \phi^{-1}(B)) \to \tilde{H}_j(A, B)$, is an isomorphism for $j < s$ and a surjection for $j = s$, then the 5-Lemma from homological algebra [13] will imply that the same conclusion holds for the second vertical map, as desired.

By the excision theorem [13], we have a commutative diagram

$$\begin{tikzcd}
\tilde{H}_j(\phi^{-1}(\sigma^s), \phi^{-1}(\partial\sigma^s)) \arrow[r, "\approx"] \arrow[d, "\phi_*"] & \tilde{H}_j(\phi^{-1}(A), \phi^{-1}(B)) \arrow[d, "\phi_*"] \\
\tilde{H}_j(\sigma^s, \partial\sigma^s) \arrow[r, "\approx"] & \tilde{H}_j(A, B)
\end{tikzcd}$$

in which the horizontal maps are isomorphisms. Thus it suffices to prove that the homomorphism $\phi_* : \tilde{H}_j(\phi^{-1}(\sigma^s), \phi^{-1}(\partial\sigma^s)) \to \tilde{H}_j(\sigma^s, \partial\sigma^s)$ is an isomorphism for $j < s$, and a surjection for $j = s$.

This will be done using the long exact sequence of the pairs $(\phi^{-1}(\sigma^s), \phi^{-1}(\partial\sigma^s))$ and $(\sigma^s, \partial\sigma^s)$.

$$\begin{tikzcd}
\tilde{H}_j(\phi^{-1}(\sigma^s)) \arrow[r] \arrow[d] & \tilde{H}_j(\phi^{-1}(\sigma^s), \phi^{-1}(\partial\sigma^s)) \arrow[r, "\partial"] \arrow[d] & \tilde{H}_{j-1}(\phi^{-1}(\partial\sigma^s)) \arrow[r] \arrow[d, "\approx"] & \tilde{H}_{j-1}(\phi^{-1}(\sigma^s)) = 0 \arrow[d, "\approx"] \\
0 = \tilde{H}_j(\sigma^s) \arrow[r] & \tilde{H}_j(\sigma^s, \partial\sigma^s) \arrow[r] & \tilde{H}_{j-1}(\partial\sigma^s) \arrow[r] & \tilde{H}_{j-1}(\sigma^s) = 0
\end{tikzcd}$$

The fourth vertical map is the isomorphism $0 \to 0$, while the third vertical map is an isomorphism by the induction hypothesis. The first vertical map is the isomorphism $0 \to 0$ for $j < s$, and is a surjection for $j = s$ (trivially, because the target group is 0), so the 5-Lemma implies the desired conclusion for $\phi_* : \tilde{H}_j(\phi^{-1}(\sigma^s), \phi^{-1}(\partial\sigma^s)) \to \tilde{H}_j(\sigma^s, \partial\sigma^s)$. ■

To complete the proof of Theorem 29, assume by way of contradiction that ϕ misses an interior point $p \in \Delta^r$, and let $\pi : \Delta^r - \{p\} \to \partial\Delta^r$ denote the continuous map which radially projects each point q of $\Delta^r - \{p\}$ to the boundary by drawing a ray from p through q and continuing until the ray hits $\partial\Delta^r$. Then we have a commutative diagram:

$$\begin{tikzcd}
\phi^{-1}(\partial\Delta^r) \arrow[r, hook, "i"] \arrow[d, "\phi"] & X \arrow[d, "\phi"] \\
\partial\Delta^r & \Delta^r - \{p\} \arrow[l, "\pi"]
\end{tikzcd}$$

Applying the functor $\tilde{H}_{r-1}$ to the above diagram we obtain

$$\begin{tikzcd}
\tilde{H}_{r-1}(\phi^{-1}(\partial\Delta^r)) \arrow[r, "i_*"] \arrow[d, "\phi_*"] & \tilde{H}_{r-1}(X) = 0 \arrow[d, "\phi_*"] \\
\mathbb{Z} = \tilde{H}_{r-1}(\partial\Delta^r) & \tilde{H}_{r-1}(\Delta^r - \{p\}) \arrow[l, "\pi_*"]
\end{tikzcd}$$

According to this diagram, the left vertical map $\phi_* : \tilde{H}_{r-1}(\phi^{-1}(\partial\Delta^r)) \to \mathbb{Z}$ factors through $\tilde{H}_{r-1}(X) = 0$, so it is the zero map. This contradicts Lemma 30, which says that it is a surjection. ■

7 Open problems

We conclude by listing some directions for research in confluent flows that we believe warrant further attention.

1. *Route in rounds.* Our focus in this paper has been on congestion minimization and demand maximization. Another natural goal is to determine the minimum number of rounds needed to route a given set of demands to a given sink, under the constraint that the routes within each round form a confluent flow of congestion 1. Formally, we would like to partition the set V of vertices into r subsets $V_1, \cdots, V_r$, such that the demands in each subset V_i can be satisfied by a confluent flow in G with congestion at most 1, and r is minimized. We have a trivial $O(\log k)$ approximation, based on the congestion minimization results; thus, $r = O(\log k)$. If G admits a splittable flow of congestion at most 1, does $r = 2$ always suffice?

2. *Remove "almost" from the title.* Tighten any of the following gaps:

   (a) H_k vs. $1 + \ln(k)$ for the worst-case ratio of splittable to confluent congestion.

   (b) $\lg(k)/2$ vs. $1 + \ln(k)$ for the optimal approximation ratio of polynomial-time algorithms for min-congestion confluent flow.

   (c) 2 vs. 3 for the optimal approximation ratio of polynomial-time algorithms for the demand maximization problem.

3. *Remove "existence" from the title.* Supply a constructive argument (ideally one which implies a polynomial-time algorithm) for finding confluent flows of congestion $C+d_{\max}$ in k-connected graphs with k sinks.

4. *Generalize the result for k-connected graphs with k sinks.* In an ℓ-connected graph with k sinks, admitting a splittable flow with congestion 1, does there exist a confluent flow with congestion $O(\log(k/\ell))$? An example which demonstrates a lower bound of $\Omega(\log(k/\ell))$ is obtained by modifying the example displayed in Figure 1 as follows. Add an edge between every pair of vertices in consecutively numbered rows as well as between the last row of round vertices and the sinks, then delete the topmost ℓ rows of vertices.

5. *Consider heterogeneous capacities.* An interesting direction for future research is to extend some or all of our results to graphs with heterogeneous capacities. (In such graphs, the congestion of a vertex or an edge is defined as the load divided by capacity.) Such questions are intrinsically more difficult than the ones we have considered so far, because it is no longer the case that the most congested vertex in any confluent flow is always one of the sinks.

6. *Study variants of confluent flows.* We are also interested in studying the notion of *edge-confluence* [2], in which all flows arriving at a vertex along an incoming edge depart the vertex on a single outgoing edge. Thus, every confluent flow (as defined and studied in this paper) is also edge-confluent; the converse, however, is not true. Another variation is to allow the flow leaving a vertex to use a constant number of outgoing edges, instead of one outgoing edge in the confluent setting. For instance, one interesting question is the following: if the graph admits a splittable flow of congestion 1, can we always achieve congestion $O(1)$ using a flow in which each vertex uses only $O(1)$ outgoing edges?

7. *Obtain improved results for special graphs.* It may be possible to obtain improved approximations for special classes of graphs, e.g., planar graphs. (Our arguments for the hardness of approximation in Section 3 do not immediately extend to planar graphs.)

References

[1] D. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1999.

[2] J. Chen, R. Rajaraman, and R. Sundaram. Meet and merge: Approximation algorithms for confluent flows. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 373–382, June 2003.

[3] Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19:17–41, 1999.

[4] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45:634–652, 1998.

[5] L. Ford, Jr. and D. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[6] L. Ford, Jr. and D. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.

[7] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[8] A. Frank. *Combinatorial algorithms, algorithmic proofs*. PhD thesis, Eötvös University, Budapest, 1976. In Hungarian.

[9] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, Feb. 1989.

[10] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.

[11] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. *Journal of Computer and System Sciences*, 67:473–496, 2003.

[12] E. Győri. On division of graphs to connected subgraphs. In *Proceedings of the Fifth Hungarian Combinatorial Colloquium*, Budapest, 1976.

[13] A. Hatcher. *Algebraic Topology*. Cambridge University Press, Cambridge, 2002.

[14] S. Khuller, B. Raghavachari, and N. Young. Designing multi-commodity flow trees. *Information Processing Letters*, 50:49–55, 1994.

[15] J. M. Kleinberg. Single-source unsplittable flow. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 68–77, Oct. 1996.

[16] A. Kumar, R. Rastogi, A. Silberschatz, and B. Yener. Algorithms for provisioning virtual private networks in the hose model. In *Proceedings of the ACM SIGCOMM Conference*, pages 135–148, 2001.

[17] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming*, 46(3):259–271, 1990.

[18] D. Lorenz, A. Orda, D. Raz, and Y. Shavitt. How good can IP routing be? Technical Report 2001-17, DIMACS, Apr. 2001.

[19] L. Lovász. A homology theory for spanning trees of a graph. *Acta Mathematica Academiae Scientiarum Hungaricae*, 30(3-4):241–251, 1977.

[20] N. Megiddo. A good algorithm for lexicographically optimal flows in multi-terminal networks. *Bulletin of the American Mathematical Society*, 83(3):407–409, May 1977.

[21] J. Wang and K. Nahrstedt. Hop-by-hop routing algorithms for premium-class traffic in diffserv networks. *ACM SIGCOMM Computer Communication Review*, 32:73–88, November 2002.

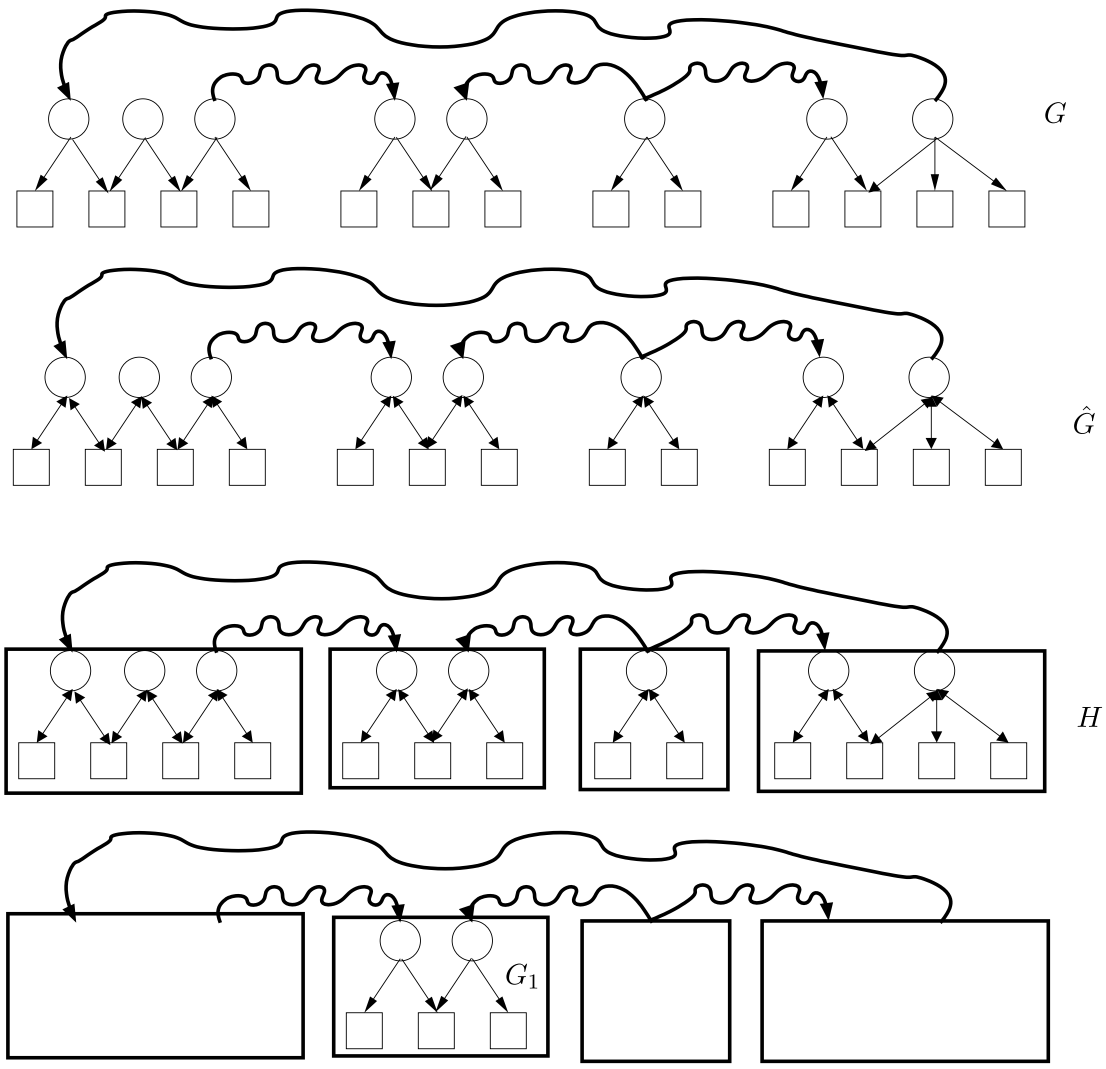

Figure 5: Construction of graph G_1 in the parsimonious sink deactivation step. The four figures represent, from top to bottom, the dag G, the graph $\hat{G}$ constructed in Algorithm CONFLT, the graph H defined in the proof of Lemma 7, and the definition of G_1. In the above figures, directed cycles of length two have been represented as bidirected edges, for convenience. Circles represent frontier nodes, squares represent sinks, and the curved lines represent collections of paths traversing nodes that are neither frontier nodes nor sinks. The rectangular boxes in the bottom two figures represent the strongly connected components of $\hat{G}$.

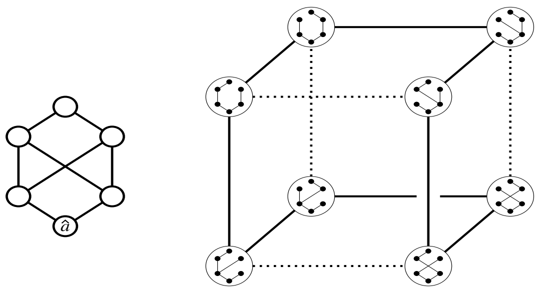

Figure 6: A graph and its arborescence complex. All edges of the graph are directed downward.