(Almost) Tight Bounds and Existence Theorems for Confluent Flows

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

Abstract

A flow is said to be confluent if at any node all the flow leaves along a single edge. Given a directed graph G with k sinks and non-zero demands on all the nodes of G, we consider the problem of determining a confluent flow that routes every node demand to some sink such that the maximum congestion at a sink is minimized. Confluent flows arise in a variety of application areas, most notably in networking; in fact, most flows in the Internet are confluent since Internet routing is destination based.

We present near-tight approximation algorithms, hardness results, and existence theorems for confluent flows. 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, 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 $(\lg k)/2$, thus resolving the polynomial-time approximability to within a multiplicative constant. We also show that a simple post-processing step following the congestion minimization algorithm yields a confluent flow with congestion at most 1 that satisfies 1/6 fraction of total demand.

We show that the gap between confluent flows and splittable flows is much smaller, if the underlying graph were 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. The proof of this existence theorem is non-constructive and relies on topological techniques introduced in [17].

[*]College of Computer and Information Science, Northeastern University, Boston MA 02115. Email:{chenj,rraj}@ccs.neu.edu.

[†]Department of Mathematics, MIT, 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 and Information Science, Northeastern University, Boston MA 02115, & Akamai Technologies, Cambridge, MA. Email:koods@ccs.neu.edu.

[¶]Department of Mathematics and School of Computer Science, McGill University, Montreal, Quebec, H3A 2A7, Canada. Email: vetta@math.mcgill.ca.

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 in a directed graph is said to be *confluent* if all the flow 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 and vice versa. 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; thus, 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.

The main focus of this paper is on finding confluent flows with small congestion. Consider a directed graph G with k distinguished nodes, referred to as *sinks*, and non-zero demands on all the nodes of G. We would like to determine 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. If we drop the confluence constraint and thus allow *splittable flows*, then a flow that minimizes maximum congestion at a sink can be obtained in polynomial time by a straightforward reduction to the maximum flow problem. On the other hand, it was shown in [1] that minimizing confluent flow congestion is MAXSNP-hard, and that an $\widetilde{O}(\sqrt{n})$ approximation is achievable for an n-node graph in the special case when all nodes have identical demands.

1.1 Our results

We present near-tight bounds on the approximability of confluent flows, and on the gap between confluent flows and splittable flows.

- 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 (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 (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 [1].

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 $(\lg k)/2$, thus resolving the polynomial-time approximability to within a multiplicative constant (Section 3.2)[1]. It is interesting to note that our lower bound is not based upon a reduction from set cover [3] and relies on a weaker precondition than that used in the set cover hardness result.

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

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 whether 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. 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 [17].

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 a postprocessing of the confluent flow obtained by the congestion minimization algorithm yields a 6-approximation to the demand maximization problem (Section 5).

Due to space constraints, we have omitted many of the proofs; they may be found in appendices A through C.

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 [4, 5]; 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 [2, 13]. The relationship between the *edge congestion* of confluent and unsplittable flows is addressed in [16], 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 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 [7] conjectured in 1975 that such partitions always exist, provided that $kC \geq n$. In fact, he made the much stronger conjecture that given sinks $s_1, \ldots, s_k$ and positive integers $n_1, \ldots, n_k$ 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 Lovasz [17] and Győry [10]. Lovasz'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 [1]. Multicommodity confluent flows are considered by [8, 14] 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 [12], 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 [16] and [18].

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 at each vertex. We denote the total demand, $\sum_{v \in V} d(v)$, by D and the maximum demand, $\max_{v \in V} d(v)$, by $d_{\max}$.

Let $S = \{s_1, \ldots, s_k\} \subseteq V$ 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 $d(v) + \text{in}(v)$.

Without loss of generality, we assume that each sink has only incoming edges. (Suppose s_i has outgoing edges, we can add a sink vertex s_i' and an edge (s_i, s_i') to G and remove s_i from S.) We also assume without loss of generality that the maximum congestion of any node of G is 1; otherwise, all the demands and flows can be scaled by the maximum congestion to satisfy this property.

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 that satisfies maximum total demand among all confluent flows that have congestion at most 1.

3 Lower bounds

In this section, we present two 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 $\frac{1}{2}\lg k$.

3.1 Confluent to splittable: H_k gap

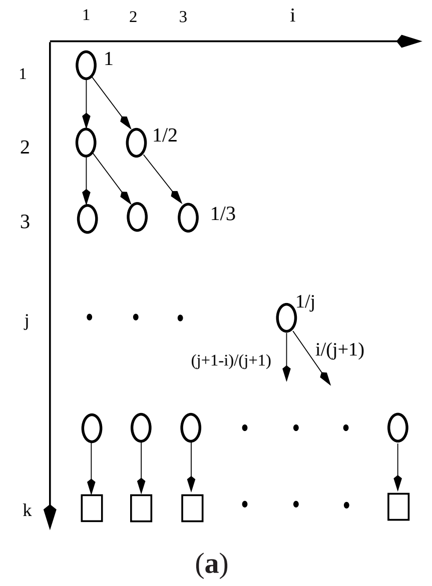
Figure 1(a) shows an instance with splittable congestion 1 but where the optimal confluent flow has congestion at least H_k. The congestion of any confluent flow is at least H_k since the node with demand 1 induces a flow of H_k into any sink that drains it. We leave it as an easy exercise for the reader to see that the flow that splits the outgoing flow at each node as shown in the figure achieves a congestion of 1.

3.2 Hardness of approximation

In [1] 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 [1] 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 [6] that it is NP-hard to determine whether or not there are vertex-disjoint 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 that $\frac{1}{2}\lg k$ 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 [9] for the edge-disjoint path problem.

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



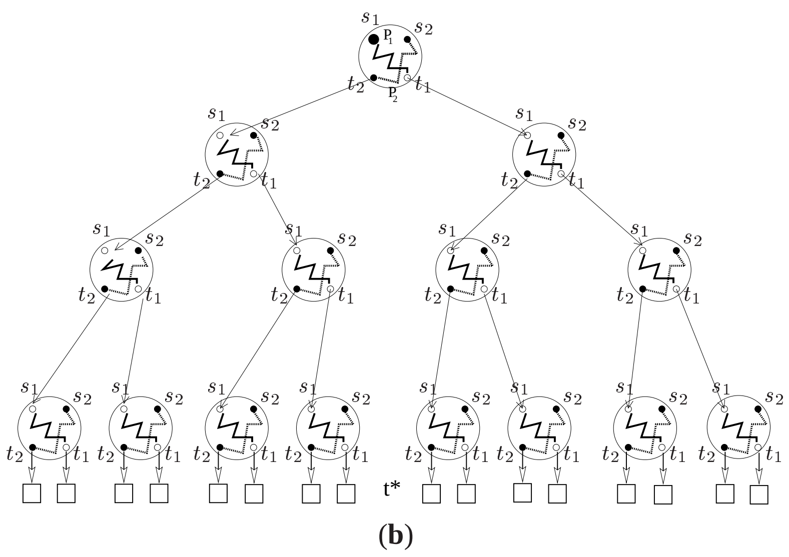


Figure 1: (**a**) Instance demonstrating an H_k gap between confluent and splittable flows. The round nodes are sources with j nodes at level j each with supply $\frac{1}{j}$. The square nodes are sinks. The flows on the outgoing edges of the ith node on level j are specified. (**b**) Instance used in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

Proof. Given G we build an auxiliary network N as follows. Take a complete binary tree T on $2^{\lceil \lg k \rceil - 1} + 1$ 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^v, t_2^{*u}), where t_1^{*u} and t_2^{*u} are sinks. Our construction is illustrated in Figure 1(b).

In addition, we give each vertex in the auxiliary network a demand. Every copy of s_2 and t_2 receives demand one. Every copy of s_1 and t_1 receives demand zero, except for s_1^r which receives 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 1(b). Since each dipath contains only two vertices of non-zero demand (and exactly one for the dipath from s_1^r to t^*), the congestion of the resultant confluent flow is exactly two. This is clearly 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 $\lceil \lg k \rceil$. Towards this end, assume that P passes through copies of G corresponding to the nodes $r = v_1, v_2, \ldots, v_{\lceil \lg k \rceil - 1}$ of T. Thus by construction P must pass through $s_1^{v_i}$, for $1 \le i \le \lceil \lg k \rceil - 1$. 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 2. 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

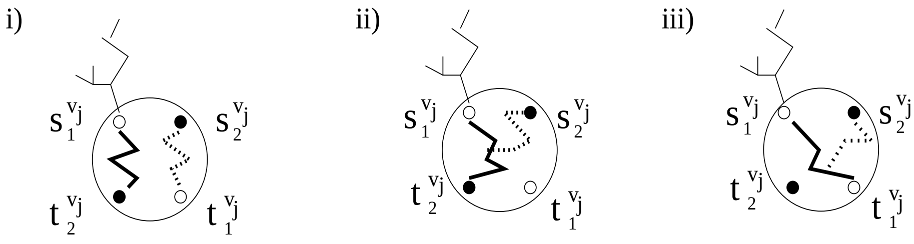


Figure 2: Routing in Networks without Disjoint Dipaths.

(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 $s_1^{v_{\lceil \lg k - 1 \rceil}}$ is $\lceil \lg k \rceil$. Observe that the number of sinks in the auxiliary network is k.

It follows that any approximation algorithm for the confluent flow problem with approximability guarantee better than $\frac{1}{2} \log k$ 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. □

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 maximum node congestion 1. The preceding flow can be determined in polynomial time using a standard maximum flow algorithm. Without loss of generality, we may assume that the given directed graph G is 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) remove an edge (often by breaking certain undirected cycles[2]) and redirect flow; (ii) aggregate a node into a sink if all of the outgoing edges of the node are to the sink; and (iii) deactivate a sink by removing all edges incident into the sink and redirecting flow. 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. 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 nodes representing the set of nodes that have been aggregated into a sink (including the sink). Any spanning forest of the edges removed during the aggregation process yields the disjoint trees $\{T_1, \ldots, T_k\}$ forming the desired confluent flow.

[2]We note that cycle-breaking is also an important component of the approximation algorithms for unsplittable flow [13, 2].

We are now ready to define our algorithm CONFLUENT. At any stage of CONFLUENT, 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.

CONFLUENT(G, S, d, f): While $V(G) \neq S$, execute the following steps:

1. **Construction of auxiliary graphs:** Construct the bipartite graph consisting of the active sinks, frontier nodes, and edges from frontier nodes to active sinks. Let $G_1, G_2, \ldots$ be the connected components of this bipartite graph. Let H denote the graph obtained by contracting each of these G_i to a single node in G (multi-edges and self-loops[3] are preserved).

2. **Breaking alternating cycles:** If any G_i contains an undirected cycle, update f so that an edge can be removed from G without increasing any node congestion. This can be done by identifying the edge with the lowest flow in the (alternating) cycle and sending an equal but opposite flow through the cycle (see Figure 3). Go back to step 1.

3. **Breaking sawtooth cycles:** If H contains a directed flow cycle (or a self-loop) C, update f so that an edge can be removed from G without increasing any node congestion. This is done as follows. Each vertex G_i in H with edges (G_j, G_i) and (G_i, G_ℓ) in C has two frontier nodes v_j and v_ℓ such that v_j (resp., v_ℓ) has a directed path to a frontier node in G_j (resp., G_ℓ). We refer to v_j (resp., v_ℓ) as the entry (resp., exit) node of G_i. Construct an undirected cycle C' in G by replacing each G_i in C by an undirected path inside G_i that connects the entry and exit nodes of G_i. The cycle C' consists of an alternating sequence of directed paths and alternating sawtooth-like paths (see Figure 3). Identify the edge in C' that has the lowest flow among all edges having the same direction as C; send an equal but opposite flow through the cycle. Remove all edges with zero flow and go back to step 1.

4. **Node aggregation:** If a frontier node v has all of its outgoing edges going into one sink s_i, mark any one of these edges, seize v into s_i and add $d(v)$ to $d(s_i)$ (see Figure 3). Go back to step 1.

5. **Sink deactivation:** Find a G_r with no outgoing edges. In G_r, find a leaf sink node s_j. Let v be the frontier node adjacent to s_j and let $s_\ell \neq s_j$ be a sink 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 (see Figure 3). Go back to step 1.

6. **Output:** Output the marked edges.

Theorem 2. *Given a splittable flow with node congestion* 1 *on a graph with* k *sinks,* CONFLUENT *finds a confluent flow that satisfes 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 $\phi(s_i) = 2^{s_i}$, 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 3 (Breaking alternating cycles). *If any G_r contains an undirected cycle, then the flow can be modified so that an edge can be removed from G without changing node congestion.*

Proof. Let C be a set of edges in G_r that forms an undirected cycle when edge directions are ignored. Since G_r is bipartite, C is an alternating cycle. Each frontier node in C has two outgoing edges while every sink node in the cycle has two incoming edges. Hence any circulation in C leaves node congestion unchanged. Consider the edge with the lowest flow in C. Step 2 of CONFLUENT sends an equal and opposite flow through the cycle and removes the edge; the congestion of every node is left unchanged. □

[3]A self-loop in H is created by an edge from one frontier node in G_i to another frontier node in G_i, for any i.

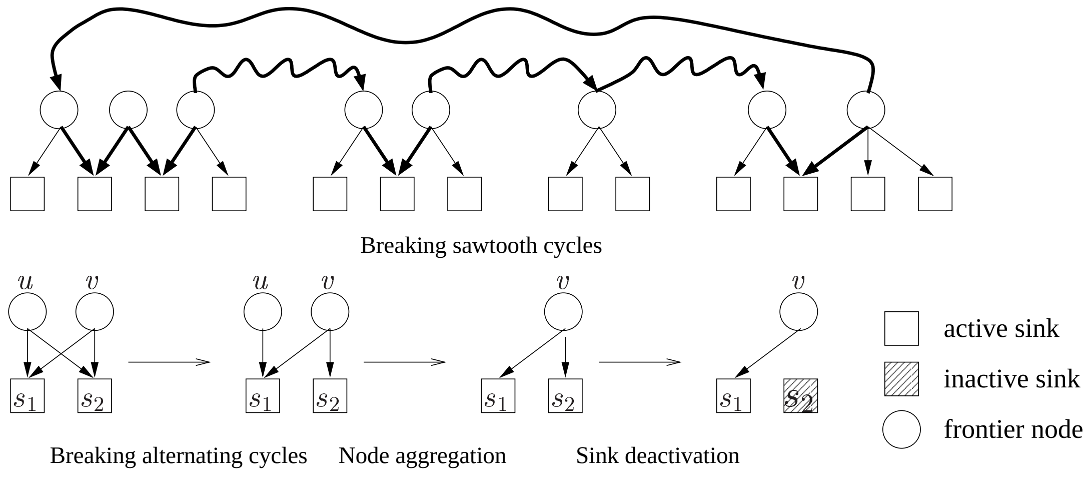


Figure 3: Illustrating steps 2, 3, 4, and 5 of CONFLUENT. In the top figure, the bold edges form a sawtooth cycle.

Lemma 4 (Breaking sawtooth cycles). *If H contains a directed flow cycle (or a self loop) C then the flow can be modified so that an edge can be removed from G without increasing node congestion.*

Proof. We expand the directed flow cycle C by expanding each G_i as indicated in step 3 of the algorithm. There are only two kinds of frontier nodes in the resulting expanded cycle: type I, which are entry nodes of the directed cycle into G_i and have one incoming edge and one outgoing edge, and type II, which have two outgoing edges. All sink nodes in the cycle have two incoming edges each; hence, any circulation leaves their congestions unchanged. Similarly the congestion of any type II frontier node is unchanged. Consider the edge e with the lowest flow among all edges having the same direction as C. Step 3 sends an equal and opposite flow through the cycle, and then removes e. The congestion of all nodes except sink nodes and type II frontier nodes only decreases, while those of the remaining nodes is left unchanged. $\square$

Lemma 5 (Node aggregation). *If a frontier node has exactly one outgoing edge then the splittable flow can be modified so that a node can be removed from G without increasing node congestion.*

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 edge (v, s_i) is removed, and the flow along all the other edges remains unchanged. $\square$

Lemma 6 (Sink deactivation). *If none of the preconditions of Lemmas 3, 4, and 5 hold then the sink deactivation step either removes an edge or deactivates a sink without increasing the potential of the resultant flow. Furthermore, the potential of the deactivated sink, if any, is no more than the potential of the flow.*

Proof. Since the precondition of Lemma 4 does not hold it follows that H is a directed acyclic graph. Consider node G_r of H with no outgoing edges in H. By the choice of G_r, it follows that no frontier node in G_r has an outgoing edge in G that is not in G_r. Furthermore, since the precondition of Lemma 3 does not hold, G_r is a tree. Finally, since the precondition of Lemma 5 does not hold, every frontier node in G_r has at least two outgoing edges in G_r. This implies that all of the leaf nodes in G_r are sink nodes.

Let s_j be a leaf sink node and let its adjacent frontier node be v; since v cannot be a leaf let $s_\ell \neq s_j$ be an adjacent sink node. 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

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

(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. $\square$

Proof of Theorem 2: We first observe that for any non-sink node CONFLUENT 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 3 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 $1 + \lg k$. $\square$

4.2 An improved upper bound

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

We now present the sink deactivation step for REFINEDCONFLUENT. As in the analysis of CONFLUENT, 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.

5. **Parsimonious sink deactivation:** Find a G_r with no outgoing edges; $S_r \leftarrow$ set of sinks in G_r.
   - **Balancing:** Redistribute the flow from frontier nodes in G_r to S_r so that the potential $\sum_{s_i \in S_r} \phi(b_i)$ is minimized. Remove any edges with zero flow.
   - Find the sink $s \in S_r$ 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 $S_r \leftarrow S_r - \{s\}$.
   - Repeat **Balancing** and then go back to step 1.

To complete the description of the parsimonious sink deactivation step, we need to specify how the step **Balancing** is implemented. The minimization problem in **Balancing** is to minimize a convex cost function subject to certain linear constraints. In Appendix A.1, we show that the unique minimum of $\sum_{s_i \in S_r} e^{b_i}$ is identical to that of any other strictly convex function, and can be obtained by a polynomial time algorithm based on max flow.

In the following, we prove that the net change in potential as a result of the parsimonious sink deactivation step is nonpositive. For any edge (v, s) in G_i from a frontier node v to 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 7. *Let v denote an arbitrary frontier node and let s denote an adjacent sink. Let β denote the smallest in-flow in G_r and $\alpha \le \beta$ 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 resultant additional demand to sinks in $T_{(v,s)}$ such that the increase in potential is at most $(\alpha/\delta)(\phi(b + \delta) - \phi(b))$, for any $\delta \in [1 - \beta, \alpha]$.*

Lemma 8. *The parsimonious sink deactivation step does not increase the potential.*

Theorem 9. *Given a splittable flow with node congestion 1 on a graph with k sinks, there exists a polynomial time algorithm that finds a confluent flow satisfying the same demand and with node congestion not exceeding $1 + \ln k$.*

Proof. Lemmas 3, 4, 5, and 8 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 $1 + \ln k$. $\square$

5 Demand maximization

In the demand maximization problem, we need to find a routable subset of the nodes of maximum total demand, requiring a confluent flow. In this section, we present a polynomial time $1/6$-approximation algorithm, i.e. it finds a confluent flow to satisfy demands of a subset of nodes that are at least $D/6$.

The demand maximization algorithm first runs CONFLUENT and obtains a set of disjoint arborescences $\{T_i\}_{i=1}^{k}$ where each T_i contains total demand b_i. We next select a subset of the nodes and satisfy their demands as follows. For each T_i, if $b_i \leq 2$, then greedily partition the nodes in T_i into groups whose total demands sum to at most $b_i/2$; select the nodes in the group with the maximum total demand and satisfy their demands using T_i. If $b_i > 2$, then search the nodes of T_i in any order and determine a subset of the nodes whose total demand is at least $1/2$. We defer the details to the appendix.

Theorem 10. *Given a splittable flow with node congestion 1 on a graph with total demand D, the algorithm above finds a confluent flow with node congestion at most 1, satisfying demands of a subset of nodes, with total demand at least $D/6$.*

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 $< C + d_{\max}$. In fact we will prove the following 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.

Theorem 11. *Let G be a directed graph with sinks $S = \{s_1, \ldots, s_k\}$, 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 $t_1, \ldots, t_k$ 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 $t_1 = \ldots = t_k = D/k < C$, we obtain a confluent flow of congestion $< C + d_{\max}$ as claimed. Theorem 11 can be seen as a weighted version of the following theorem.

Theorem 12 (Lovasz, 1977 [17]). *Let G be a digraph, $v_1, \ldots, v_k \in V(G)$ and assume that G is k-connected to S. Let, furthermore, k positive integers $n_1, \ldots, n_k$ be given whose sum is $|V(G)|$. Then G contains k vertex-disjoint arborescences $A_1, \ldots, A_k$, such A_i is rooted at v_i and $|V(A_i)| = n_i$.*

It is worthwhile to introduce here *nearly confluent* flows: these are splittable flows where the flow may split only at nodes with no incoming flow. Theorem 11 is a consequence of the following two facts about nearly confluent flows. First, if we allow nearly confluent flows, then the congestions at the sinks can be prescribed arbitrarily:

Theorem 13. *Let G be a directed graph with sinks $S = \{s_1, \ldots, s_k\}$, demands $d : V \to \mathbb{R}_+$ and total demand D. 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.*

Second, a nearly confluent flow may be rounded to a confluent flow without increasing the maximum edge congestion by too much.

Theorem 14. *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 for all i.*

We'll obtain Theorem 11 using the topological techniques which appear in the proof of Theorem 12. See [10] for a combinatorial proof of Theorem 12 for k-connected undirected graphs.

The arborescence complex of a directed graph. In [17] 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 cellular complex whose vertices 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.

Definition 15. 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 $\hat{a}$.

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 K is the subset of the cube where this criterion holds; it is an octagon consisting of eight edges of the cube.

The fundamental fact about the topology of arborescence complexes is the following theorem from [17].

Theorem 16 ([17], 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 [17], the importance of Theorem 16 is that it enables us to apply a generalized intermediate value theorem to obtain a near-arborescence with desired properties.

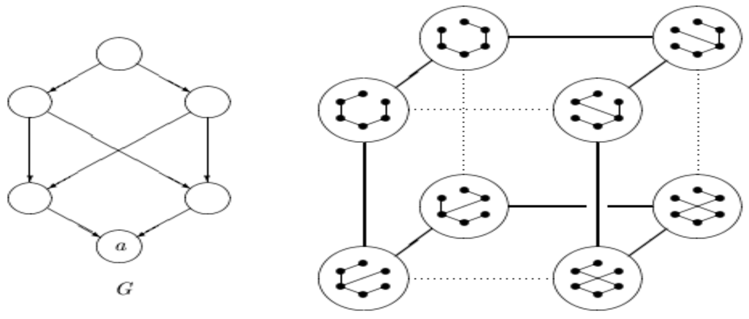


Figure 4: An arborescence complex

Proof of Theorem 13: 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 constitute a DAG in order for this flow to be well-defined. If F is a near-arborescence, it is automatic that this edge set is a DAG.) 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 we will be done if we can find 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 deferred to the Appendix.

Theorem 17. *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 apply Theorem 17 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 $v \neq a$ can be separated from $\hat{a}$ by removing fewer than $s+1$ vertices, so Theorem 16 ensures the vanishing of the required homology groups. □

References

[1] 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*, June 2003.

[2] Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 290–299, November 1998.

[3] U. Feige. A threshold of $\ln n$ for approximating set cover. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 314–318, May 1996.

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

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

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

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

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

[9] 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. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 19–28, May 1999.

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

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

[12] S. Khuller, B. Raghavachari, and N. Young. Designing multi-commodity flow trees. In F. K. H. A. Dehne, J.-R. Sack, N. Santoro, and S. Whitesides, editors, *Proceedings of the 3rd Workshop on Algorithms and Data Structures*, volume 709 of *Lecture Notes in Computer Science*, pages 433–441, Montréal, Canada, Aug. 1993. Springer.

[13] 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.

[14] 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 2001 Conference*, volume 31 of *Computer Communication Review*, pages 135–148, Aug. 2001.

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

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

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

[18] J. Wang and K. Nahrstedt. Hop-by-hop routing algorithms for premium-class traffic in diffserv networks. In *Proc. of IEEE INFOCOM 2002*, New York, NY, June 2002.

A Proofs for Section 4.2

A.1 Convex minimization

In this section we consider the convex program obtained in trying to minimize the potential function over the sinks. We characterize the structure of the minimizing point and use the structure to argue that in fact the point of minimization is independent of the cost function. We conclude by showing that the elucidated structure also enables us to find this point of minimization in polynomial time using max flow as a subroutine.

Consider the bipartite graph formed by the frontier nodes $\{f_i\}$ and the sinks $\{s_j\}$. Let x_{ij} represent the flow on the edge from frontier node f_i to sink s_j. Let y_j represent the total flow into sink s_j. Let d_i be the supply at frontier node f_i and d'_j be the intrinsic supply at sink s_j. Let $C(y)$ be a nonnegative, increasing and strictly convex function. Consider the following program:

$$\begin{aligned}
& \min_j C(y_j) \\
\text{s.t.} & \\
\forall i, \quad & \sum_j x_{ij} \geq d_i \\
\forall j, \quad & y_j \geq \sum_i x_{ij} + d'_j \\
\forall i,j, \quad & x_{ij} \geq 0
\end{aligned}$$

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 minimum. We also refer to this convex objective function as the potential function. Note that anytime we have a setting of the variables such that all the constraints are tight and the y values are all equal then we are at the global minimum since the sum of the supplies (intrinsic and otherwise) is a lower bound on the sum of the y's. Observe also that, though many different settings to the x variables may induce the same set of y variables, the y values are a continuous function of the x values.

We define the following "balance" operation — if a frontier node has edges with nonzero flow to two sinks with different y values then by shifting flow (i.e. x value) from the sink with larger y value to the sink with lower y value the value of the potential function is reduced. This follows from the convexity of C. We also refer to the operation as balancing the frontier node in question. Note also that the potential function is never reduced by increasing the flow out of a frontier node and this follows from the fact that C is increasing.

Observe that a setting of the x variables determines the value of the convex program. If a setting of the convex program does not allow for further balancing then we say that the configuration is a local minimum with respect to balancing. Observe that if there is a setting of y values to be equal and the constraints to be tight then that is the minimum.

Theorem 18. *Let X be a setting of x variables that satisfies the convex program and is a local minimum with respect to balancing. Then the partition of the sinks based on their y values induces a corresponding partition of the frontier nodes. Every frontier node sends flow only to sinks with the same y value. Further a frontier node corresponding to a specific y value will have no edges to sinks with a lower y value and will have zero flow on edges to sinks with higher y value.*

Proof. Consider a frontier node — it cannot send flow to two sinks with differing y value otherwise it could be balanced. Hence we get a natural partition. Observe that a node could not have an edge to a sink with lower y value for otherwise it could be balanced. Similarly it could not have any flow on an edge to a node with higher y value else it could be balanced. □

The above structure theorem says that at a local minimum with respect to balancing the original bipartite graph can be thought of as being partitioned into disjoint graphs and in each of these graphs the natural restriction of the original convex program achieves its global minimum. The original convex program is the sum of its restrictions.

Theorem 19. *A local minimum with respect to balancing is a local minimum.*

Proof. Consider a local minimum with respect to balancing defined by the y values. Pick any x values that induce these y values. the strategy of this proof is to show that for any perturbation π to the x values the potential function is not reduced. Since the y values are a continuous function of the x values it follows that there is no perturbation to the y values that reduces the potential and hence we are at a local minimum.

We are left to show that there is no perturbationπ to the x values that reduces the potential. The argument is by contradiction. Suppose not. Consider an π that reduces potential. If it increases the outflow of any frontier node beyond its supply then we can reduce this outflow and hence the potential and consider the corresponding perturbation. In other words if there exists a potential reducing π then there is one that holds frontier node outflows constant. By the above structure theorem edges from frontier nodes to sinks with higher y values carry no flow. If an π sends flow along such an edge then observe that by reducing the flow along this edge and sending it to a sink in the same partition we reduce the potential further. Hence there exists an π that only sends flow along the edges where the x values are non-zero. But in any partition we know we are at a global minimum so any π that holds frontier node outflows constant cannot reduce potential. Hence contradiction. $\square$

Corollary 20. A local minimum with respect to balancing is a global minimum.

Theorem 21. *The point of minimization is independent of C.*

Proof. Suppose not. Let C_1 and C_2 be two potential functions with different points of minimization, y^1 and y^2. We know that each of these points is the unique global minimum and local minimum with respect to balancing under the respective potential functions. Since y^2 is not a point of minimization under C_1 it must be capable of further balancing, but since the operation of balancing is independent of C, this means that y^2 is capable of further balancing under C_2 — a contradiction. $\square$

We now present an algorithm to find the point of minimization using max flow as a subroutine. The algorithm will be polynomial time but it will not be strongly polynomial since it will utilize binary search.

Algorithm: Create a universal sink and connect all the sink nodes s_j to it with edges of the same capacity. Run a binary search on this capacity to determine the largest value such that all these edges are saturated. Partition the sinks into two based on what happens when the capacity is increased infinitesimally beyond this point - those whose edges continue to be saturated and those whose edges are not. It is easy to see that this induces a partition of the frontier nodes along the lines of our structure theorem above. Remove the unsaturated partition and recurse on the saturated partition.

It is an easy exercise to repeat the arguments above to see that the resulting partition and corresponding x values is a local minimum with respect to balancing.

A.2 Nonpositive net change in potential

Proof of Lemma 7: The proof is by induction on the number of sinks in G_r. The assumptions about G_r require that there be at least two sinks in G_r. So, for the induction base, we consider the case when G_r 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 $\alpha \leq \beta$, $[1-\alpha, \beta]$ 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 a factor of 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 $1 \le \ell \le j$, we select $\alpha_\ell \le f(v_\ell, s)$ 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 $\delta \ge 1 - \beta$.

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 $\alpha_\ell \le \alpha$ and $\alpha \le \beta \le \beta_\ell$, it follows that $\alpha_\ell \le \beta_\ell$. By the 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 $\beta_\ell \ge \beta$ and $\delta \ge 1-\beta$, we obtain that $\delta \ge 1 - \beta_\ell$. We consider two cases. If $\delta \le \alpha_\ell$, 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

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

the last inequality following from Lemma 22 below 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 and the proof of the lemma. □

Proof of Lemma 8: Let s^* denote a sink in G_r with minimum in-flow. 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 $1 \le \ell \le j$. Let $s_\ell \neq s^*$ denote an arbitrary sink adjacent to v_ℓ. (Since every frontier node in G_r has at least two adjacent nodes, 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

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

by Lemma 22.

We now consider the case $\alpha_\ell \ge 1 - \alpha$. Let β_ℓ denote the minimum in-flow of a sink node in $T_{(v_\ell, s_\ell)}$. Since $\beta_\ell \ge \alpha$, it follows that $1 - \alpha \ge 1 - \beta_\ell$. We invoke Lemma 7, 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

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

(In the penultimate step, we use the inequality $e^{\alpha-1} \ge 1+\alpha-1=\alpha$.) $\square$

Lemma 22. *For $0 \le x \le 1$, 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

$$\begin{aligned}\frac{e^x}{x}-\frac{e^x}{x^2}+\frac{1}{x^2} &= \frac{e^x(x-1)+1}{x^2}\\ &= \frac{(x-1)/e^{-x}+1}{x^2}\\ &\ge 0,\end{aligned}$$

since $e^{-x} \ge 1-x$ and $x \le 1$. $\square$

B Proofs for Section 5

We first give a complete description of the demand maximization algorithm. The algorithm first runs CONFLUENT and obtains a set of disjoint arborescences $\{T_i\}_{i=1}^k$ where each T_i contains total demand b_i. We next select a subset of the nodes and satisfy their demands as follows. For each T_i, if $b_i \le 2$, then greedily partition the nodes in T_i into groups whose total demands sum to at most $b_i/2$; select the nodes in the group with the maximum total demand and satisfy their demands using T_i. If $b_i > 2$, then search the nodes of T_i in any order and do the following.

1. $\Delta_i \leftarrow 0$.
2. While $\Delta_i < 1/2$ do
   (a) Visit the next node v.
   (b) If $d(v) \le 1/2$, 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 23. *In* CONFLUENT, *when the congestion of any sink changes, exactly two sinks are involved. Suppose they are s_x, s_y and w.l.o.g. assume $b_x \ge b_y$. One of the following happens.*

1. *b_x decreases by δ, causing b_y to increase by δ; $b_y+\delta < b_x-\delta$; no one gets deactivated;*
2. *b_y decreases by δ, causing b_x to increase by δ; $b_y-\delta+2 \ge b_x+\delta$; b_y gets deactivated;*
3. *b_x decreases by δ, causing b_y to increase by δ; b_x gets deactivated.*

Proof. During CONFLUENT, congestion of any virtual sink may change only in step 5. When s_y is the leaf sink and $b_y + f(v, s_x) < b_x - f(v, s_x)$, we have case 1. When s_y is the leaf sink and $b_y + f(v, s_x) \geq b_x - f(v, s_x)$, we have case 2, where since $2 \geq 2\delta + 2f(v, s_x)$ we have $b_y - \delta + 2 \geq b_x + \delta$. When s_x is the leaf sink, we have case 3. □

Lemma 24. *When* CONFLUENT *terminates*

$$\sum_{i:b_i \leq 2} b_i + \sum_{i:b_i > 2} 1 \geq \frac{D}{2}$$

Proof. we first show:

$$\sum_{i:b_i > 2} (b_i - 2) \leq \sum_{i:b_i \leq 2} b_i$$

We plot $\{b_i\}_{i=1}^k$ as a histogram. The congestion of the sinks may change during the execution of CONFLUENT, specifically, in step 5. But the total area of the histogram is always D. Divide the xy-plane of the histogram into two sections by the $y = 2$ 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 in the upper section. A increases only when some area moves from the lower section to the upper section. But whenever this happens, there is some sink deactivated and the corresponding b_i, which is less than 2, remains untouched thereafter. Denote by B the sum of b_i of the set of such deactivated virtual sinks. $B \leq \sum_{i:b_i \leq 2} b_i$. Notice that initially $A = B = 0$. So we only need to show that whenever A increases, B increases by at least the same amount, i.e., a b_i of at least the same amount gets deactivated. This can be verified by checking each possibility where A increases.

Look at any moment when A increases. Denote the increment of A by δ_A, the increment of B by δ_B.

- Case 2 of Lemma 23.
  - $2 \leq b_y \leq b_x$. $\delta_A = 2 - (b_y - \delta)$, $\delta_B = b_y - \delta$. $\delta_B - \delta_A = 2(b_y - \delta - 1) \geq 0$.
  - $b_y < 2 < b_x$. $\delta_A = \delta$, $\delta_B = b_y - \delta$. $\delta_B - \delta_A = b_y - 2\delta \geq b_x - 2 \geq 0$ (since $b_y - \delta + 2 \geq b_x + \delta$).
  - $b_y \leq b_x \leq 2$. $\delta_A = b_x + \delta - 2$. $\delta_B = b_y - \delta$. $\delta_B - \delta_A = b_y - \delta + 2 - b_x - \delta \geq 0$ (since $b_y - \delta + 2 \geq b_x + \delta$).
- Case 3 of Lemma 23.
  - $2 \leq b_y \leq b_x$. $\delta_A = \delta - (b_x - 2)$. $\delta_B = b_x - \delta$. $\delta_B - \delta_A = 2(b_x - \delta - 1) \geq 2(2 - \delta - 1) \geq 0$.
  - $b_y < 2 < b_x$. $\delta_A = (b_y + \delta - 2) - (b_x - 2) = b_y + \delta - b_x$. $\delta_B = b_x - \delta$. $\delta_B - \delta_A = 2(b_x - \delta) - b_y \geq 2(2 - \delta) - b_y \geq 2 - b_y \geq 0$.
  - $b_y \leq b_x \leq 2$. $\delta_A = b_y + \delta - 2$. $\delta_B = b_x - \delta$. $\delta_B - \delta_A = (b_x - b_y) + 2(1 - \delta) \geq 0$.

Now combine $D = \sum_{i:b_i>2}(b_i - 2) + \sum_{i:b_i>2} 2 + \sum_{i:b_i \leq 2} b_i$ with $\sum_{i:b_i>2}(b_i - 2) \leq \sum_{i:b_i \leq 2} b_i$ to get the claim. □

Now we are ready to prove Theorem 10. **Proof of Theorem 10:** 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 $b_i \leq 2$, 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 $b_i/4$, otherwise d_1 and d_2 could have been combined. Since $d_1 + d_r > b_i/2$, it follows that $r \leq 3$ (otherwise $d_1 + d_2 + d_3 + d_r > b_i$). Therefore, $d_r \geq b_i/3$, i.e. at least

$b_i/3$ is satisfied; totally $\sum_{i:b_i\le 2} b_i/3$ is satisfied. For T_i with $b_i > 2$, at least $\Delta_i \ge 1/2$ demand is satisfied; totally at least $\sum_{i:b_i>2} 1/2$ is satisfied. From Lemma 24:

$$\sum_{i:b_i\le 2} \frac{b_i}{3} + \sum_{i:b_i>2} \frac{1}{2} \ge \frac{1}{3}\left(\sum_{i:b_i\le 2} b_i + \sum_{i:b_i>2} 1\right) \ge \frac{D}{6}$$

the algorithm satisfies at least $D/6$ demand in total. □

C Proofs for Section 6

Proof of Theorem 14: We describe an algorithm for rounding f to $\tilde{f}$, in the spirit of [15]. Delete all edges e with $f(e) = 0$ from G. Let $U \subseteq V(G)$ denote the set of nodes with outdegree > 1; we'll call these the *undecided nodes.* The algorithm will select undecided nodes one at a time, and will re-route all the flow from the selected node along one of its outgoing edges. It follows that the congestion 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*. As long as G contains an undirected cycle which does not pass through $\hat{a}$, we can modify f by finding an edge on this cycle with minimum flow value, sending an equal amount of flow around the cycle in the opposite direction, 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 edge adjacent to $\hat{a}$. After finitely many such operations it will be the case that every undirected cycle in G passes through $\hat{a}$.

Now let S denote the set of neighbors of $\hat{a}$. We claim there exists a vertex $s \in S$ with only one undecided vertex upstream from it. This can be proved by induction on $|U|$. If $|U| = 1$ there is nothing to prove. If $|U| > 1$, let $E_0 \subseteq E$ denote the set of active edges which are not incident to $\hat{a}$. These are the edges of a forest. Delete an edge e of E_0 to obtain a forest of at least two components, with at least one undecided vertex in each component. By the induction hypothesis, in each component there is at least one element of S with only one undecided vertex upstream. Inserting edge e again, the element of S downstream from e may now have more than one undecided vertex upstream; however this leaves at least one element of S which still has only one undecided vertex upstream.

Having found a vertex $s \in S$ 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 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 and f has been rounded to a confluent flow.

Note that if e is an edge incident to $\hat{a}$, the congestion of e only increases in the step where e 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. □

Before proving Theorem 17, we explain why we refer to it 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 the standard form of the intermediate value theorem.

Proof of Theorem 17: Suppose we are given a map $\phi : X \to \Delta^r$ satisfying the homological vanishing criterion of Theorem 17, 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 25. *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'll prove, by induction, the following much 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 $\phi_* : \tilde{H}_j(\phi^{-1}(A)) \to \tilde{H}_j(A)$ is an isomorphism for $j < s$ and a surjection for $j = s$. In the base case, $s = 0$, $\tilde{H}_0$ is a free abelian group whose rank is one less than the number of connected components; 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 $s > 1$, 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 [11] will imply that the same conclusion holds for the second vertical map, as desired.

By the excision theorem [11], 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 $\phi_* : \tilde{H}_j(\phi^{-1}(\sigma^s), \phi^{-1}(\partial\sigma^s)) \to \tilde{H}_j(\sigma^s, \partial\sigma^s)$ is an isomorphism for $j < s$, 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 $\phi_* : \tilde{H}_j(\phi^{-1}(\sigma^s), \phi^{-1}(\partial\sigma^s)) \to \tilde{H}_j(\sigma^s, \partial\sigma^s)$. $\square$

To complete the proof of Theorem 17, 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 25, which says that it is a surjection. $\square$