2015 IEEE 56th Annual Symposium on Foundations of Computer Science

Polylogarithmic Approximations for the Capacitated Single-Sink Confluent Flow Problem

F. B. Shepherd / Adrian Vetta
McGill University
Montreal, Canada
bruce.shepherd@mcgill.ca / adrian.vetta@mcgill.ca

Gordon T. Wilfong
Bell Labs, Alcatel-Lucent
Murray Hill, NJ
gtw@research.bell-labs.com

Abstract

A single-sink *confluent flow* is a routing of multiple demands to a sink r such that any flow exiting a node v must use a single arc. Hence, a confluent flow routes on a tree within the network. In uncapacitated (or uniform-capacity) networks, there is an $O(1)$-approximation algorithm for demand maximization and a logarithmic approximation algorithm for congestion minimization [6].

We study the case of capacitated networks, where each node v has its own capacity $\mu(v)$. Indeed, it was recently shown that demand maximization is inapproximable to within polynomial factors in capacitated networks [20]. We circumvent this lower bound in two ways. First, we prove that there is a polylogarithmic approximation algorithm for demand maximization in networks that satisfy the ubiquitous *no-bottleneck assumption* (NBA). Second, we show a bicriteria result for capacitated networks without the NBA: there is a polylog factor approximation guarantee for demand maximization provided we allow congestion 2.

We model the capacitated confluent flows problem using a multilayer linear programming formulation. At the heart of our approach for demand maximization is a rounding procedure for flows on multilayer networks which can be viewed as a proposal algorithm for an extension of stable matchings. In addition, the demand maximization algorithms require, as a subroutine, an algorithm for approximate congestion minimization in a special class of capacitated networks that may be of independent interest. Specifically, we present a polylogarithmic approximation algorithm for congestion minimization in monotonic networks – those networks with the property that $\mu(u) \leq \mu(v)$ for each arc (u, v).

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.51

I. Introduction

The confluent flow problem was introduced by Chen, Rajaraman and Sundaram [7] as a model for understanding the impact of next-hop routing protocols on node congestion in a single-sink network $G = (V, A)$. There are $|V| = n$ nodes and each node v has a capacity $\mu(v)$ and has a commodity (demand) of size $d(v) \geq 0$ to be routed to a fixed destination (IP address) r.[1] The next-hop routing constraint requires that any two demands that meet at a node must then traverse identical paths to r. As a result, the support of the multiflow corresponding to the routing is an arborescence rooted at r. Such a flow, with the property that all the flow exiting each v must use a single arc, is termed a *confluent flow*. The congestion of a node v is the flow out of v divided by its capacity. The congestion of a confluent flow is then the maximum congestion at any node, and this induces the two natural optimization problems that have been studied in the literature:

- `Demand Maximization`: Find a subset of the commodities of maximum total demand that can be routed via a confluent flow with congestion at most one.
- `Congestion Minimization`: Find a confluent flow of minimum congestion that routes every commodity.

There is a strong understanding of these problems in the special case of uncapacitated networks (that is, networks with uniform capacity $\mu(v) = 1$, for each node v). Specifically, Chen et al. [6] proved:

Theorem I.1. *[6] In single-sink uncapacitated networks, the confluent flow problem has a 3-approximation algorithm for demand maximization and $O(\log n)$-approximation algorithm for congestion minimization.* □

Furthermore, there are hardness bounds within a constant factor of these upper bounds. Chen et al. [6] explicitly raised the question of capacitated networks, but there has been little progress on this question. An exception is an FPTAS for demand maximization in capacitated graphs of bounded treewidth, by Dressler and Strehler [13]. Some explanation for this lack of success can be found in the following recent hardness results.

Theorem I.2. *[20] In capacitated single-sink instances, the demand maximization confluent flow problem cannot be approximated to within a factor $O(m^{\frac{1}{2}-\epsilon})$, for any $\epsilon > 0$, unless $P = NP$. (This holds even if the demand spread $\frac{d_{max}}{d_{min}} = 1 + \Delta$, where $\Delta > 0$ is arbitrarily small.)* □

Theorem I.3. *[20] In capacitated single-sink instances, the (strong) congestion minimization confluent flow problem cannot be approximated to within a factor $O(m^{.5-\epsilon})$, for any $\epsilon > 0$, unless $P = NP$.* □

Here a *strong* routing algorithm requires that every demand $d(v)$ routes only through nodes of capacity at least $d(v)$. In contrast a *weak* routing algorithm allows a demand to go through lower capacity nodes. Weak routing algorithms may seem unnatural but, for the congestion minimization problem, they are admissible because we implicitly boost the capacity of each node by our approximation guarantee. This distinction between strong and weak routings appears fundamental (see the discussion in Section II and Theorem I.7).

Interestingly, the hardness bounds in Theorems I.2 and I.3 also apply to the unsplittable flow problem [20]. At first thought, this may sound odd because there are well-known constant factor approximations for congestion minimization and demand maximization in single-sink unsplittable flow problems; see Kleinberg [16] and Dinitz et al. [11]. However, their results only apply given the *no-bottleneck assumption* (NBA) which states that every demand is smaller than every node capacity, that is $d_{\max} \leq \mu_{\min}$. This observation motivates us to examine instances of the capacitated confluent flow problem in two distinct cases. Namely, we analyse separately those networks that satisfy the no-bottleneck assumption and those that do not. We remark that, unlike for capacitated single-sink unsplittable flow, it is not possible to improve upon logarithmic upper bounds for demand maximization even when we make the NBA.

[1] The realistic setting where nodes may have multiple demands can be handled by standard techniques; we omit the details here.

Theorem I.4. *[20] In single-sink capacitated instances that satisfy the* NBA*, the demand maximization confluent flow problem cannot be approximated to within a factor $O(\log^{1-\epsilon} n)$, for any $\epsilon > 0$, unless $P = NP$.* □

A. Our Results

Our main result is that the polynomial lower bound of Theorem I.2 does not apply to networks with the NBA. Specifically, for demand maximization there is a polylogarithmic approximation algorithm.

Theorem I.5. *In single-sink capacitated instances that satisfy the* NBA*, there is a $O(\log^6 n)$ approximation algorithm for the demand maximization confluent flow problem.*

Combined with Theorem I.4, this shows that polylogarithmic approximation is tight. As a straightforward corollary, we also obtain

Corollary I.6. *In single-sink capacitated instances that satisfy the* NBA*, every demand can be routed on confluent flows in $O(\log^7 n)$ rounds.*

Now consider the general case. Here we can still circumvent the polynomial lower bound of Theorem I.2 *provided* we slightly relax the capacity constraints. In particular, we have a bicriteria approximation algorithm:

Theorem I.7. *In single-sink capacitated instances, there is an $O(\log^6 n)$ approximation algorithm for the demand maximization confluent flow problem if congestion 2 is allowed.*

Our algorithms begin with a multilayer graph relaxation that applies also to single-sink unsplittable flows. Because, confluent flows are themselves unsplittable, our bicriteria result also applies to the single-sink unsplittable flow problem. Whilst the unsplittable flow problem without the NBA has been studied on special classes of network such as paths [2] and trees [4], we believe this gives the first approximation guarantee for general networks without the NBA, but restricting to single-sink demands.

Corollary I.8. *In single-sink capacitated instances, there is a $O(\log^6 n)$ approximation algorithm for the demand maximization unsplittable flow problem if congestion 2 is allowed.*

Observe that Theorem I.7 and Corollary I.8 mirror a similar phenomenon that occurs in the maximum edge-disjoint paths problem; see the celebrated work of Chuzhoy [9] and subsequent improvements [10], [3]. Furthermore, we remark that our approximation algorithms for demand maximization rely upon a polylogarithmic approximation algorithm for congestion minimisation in a special class of capacitated networks that satisfy a monotonicity property (see Section III).

B. Overview of the Algorithms

We now present an overview of the polylog approximation algorithm for the demand maximization confluent flow problem in capacitated networks under the no-bottleneck assumption.

To begin, we preprocess the instance. First, observe that demands of size at most $\frac{d_{\max}}{2n}$ can contribute at most half of the value of the optimal solution. Thus we may remove these demands without affecting the resultant approximation guarantee. Second, we may assume that $\mu_{\max} \leq n \cdot d_{\max}$ as any spare capacity above that level is superfluous. Third, we round up each demand and each capacity to the nearest power of two. Since $d_{\max} \leq u_{\min}$, the no-bottleneck assumption is maintained after this rounding. Observe that we now have $O(\log n)$ groups of distinct demand sizes and $O(\log n)$ distinct capacity sizes. We will run our algorithm separately on each distinct demand group, and output the best solution we obtain from amongst them. Because, in the optimal solution, one of the demand size groupings must contribute at least a logarithmic fraction of the total demand, this will lose only a logarithmic factor in the approximation guarantee. Consequently, we have an instance with uniform demand sizes. Furthermore, by scaling, we may assume that every demand in our instance is exactly $2^0 = 1$. Hence all the data in our processed instance is integral.

In the processed instance, we will ultimately show how to find an integral confluent flow $\hat{f}$ with a bicriteria approximation guarantee. Specifically, it will route a $\Omega(1/\log n)$ fraction of optimal demand with congestion at most $O(\log^4 n)$. To prove Theorem I.5, we then apply the following result of Chekuri et al. [5] to the flow $\hat{f}$.

Theorem I.9. *[5] Let T be an edge-capacitated tree instance with the* NBA*. Suppose there is a fractional flow that routes a fraction $\gamma_i \in [0,1]$ of each commodity i. Then a subset of the items can be found, in polynomial time, that feasibly routes a total demand of $\Omega(\sum_i \gamma_i d(i))$ on T.* □

To see this, note that we can scale our integral flow $\hat{f}$ by $\frac{1}{log^4 n}$. This produces a fractional flow f with $\gamma_i = \Omega(1/\log^4 n)$ for each commodity, but only routes an $\Omega(1/\log^4 n)$ fraction of the demand routed by $\hat{f}$. Combined with the additional $\log n$ lost by the grouping process this produces the $O(\log^6 n)$ approximation guarantee of Theorem I.5.

So the difficult part is to obtain the above bicriteria confluent flow $\hat{f}$. The natural way to attempt this would be to round the fractional network flow on $G = (V, A)$ (as is done for the uncapacitated confluent flow problem [6]). However, such an approach is doomed to fail; we discuss the reasons for this in detail in Section II. Instead, we round a network flow on a *multilayer* extension H of the network G. In each layer ℓ of H there is a copy v^ℓ of a node v with capacity 2^ℓ. For each arc $(u, v) \in G$, there is a (*vertical*) arc (u^ℓ, v^ℓ) in H if both u and v have capacity at least 2^ℓ in G. Furthermore, H contains each (*horizontal*) arc $(v^\ell, v^{\ell+1})$. It can be seen that any confluent routing in G maps to a confluent flow in H of similar congestion. The formulation using H is strong, however, in the sense that some fractional flows in G are not feasible in H. This is all described in Section IV.

The cost of using the multilayer graph H is that its confluent flows need not correspond to confluent flows in G. This is because different copies v^ℓ and $v^{\ell'}$ of v may *conflict* – they have outgoing vertical arcs to different out-neighbours in their respective layers. Thus a confluent flow in H may correspond to a routing in G with logarithmic out-degrees since, by the preprocessing, there are $O(\log n)$ layers in H. We will address this problem with the layered graph momentarily. First, let's understand why using the layered approach is advantageous. Observe, that the layered graph H is *monotonic* in the sense that $\mu(u) \leq \mu(v)$ for each arc (u, v). This is important because for monotonic networks, there is a polylogarithmic approximation algorithm for the minimum congestion capacitated confluent flow problem; we present this result in Section III. Indeed the approximation guarantee for congestion is the factor $O(\log^4 n)$ desired by our bicriteria confluent flow.

It remains to deal with the conflicts arising from the confluent flow in the layered graph H. We explain how to do this in Section V. There we show how to extract a logarithmic fraction of the demands that can be rerouted to induce a conflict-free confluent flow in the layered graph H and, thus, a confluent flow in the original network G. Applying Theorem I.9 as described gives our polylogarithmic approximation algorithm for maximum confluent flow in networks with the NBA.

In networks without the NBA we apply the same approach to obtain the bicriteria result, Theorem I.7. The only difference is that, without the NBA, when we round the demands and capacities a demand $d(i)$ may now be allowed to route through a node j of smaller capacity provided $d(i)$ and $\mu(j)$ both round up to the same power of two. The ultimate consequence will be that we will need a factor *two* congestion to route our demands. However, once the rounding has been applied and we focus upon a single demand group, the processed instance now satisfies the no-bottleneck assumption! The arguments for the NBA case can then be applied as before.

C. Related Work

An interesting special case of confluent flows was already studied by Lovász [17]. He shows that if a directed graph is k-connected to a subset $S = \{a_1, a_2, \ldots, a_k\}$ of nodes, then for any positive integers n_i with $\sum_{i=1}^{k} n_i = n$, there exist node-disjoint arborescences $A_1, A_2, \ldots, A_k$ such that A_i is rooted at a_i and contains n_i nodes. The implication for confluent flows to a sink node r is as follows. Suppose the digraph G

is k-connected to k neighbhours of t. Then by taking the n_i's to be an equitable partition of n, one deduces that a unit demand can be routed from each node to t by a confluent flow with the optimal node congestion of $\lceil n/k \rceil$. This result is extended to non-unit demands in [6].

As mentioned, the only positive results on capacitated confluent flows concern outerplanar and bounded treewidth networks [14], [13] whereas, on the negative side, [20] shows that demand maximization is inapproximable to polynomial factors. Some applications of confluent flows have also arisen. The maximum confluent flow problem was used in [18] to solve rooted clustering problems which are used in maximum disjoint path algorithms. Recently confluent flows were also employed to show the linear dependence between balanced separators and treewidth [14]. There have been several applications to networking including [1], [8]. The general confluent flow problem with node capacities was actually mooted in [19] as a model for understanding interdomain routing behaviour on the Internet. We build intuition for the conflict-free confluent flow algorithm by discussing the conflict-free disjoint-path problem first. This problem on the 2-layer graph can actually be solved (in exponential time) using a stable matching algorithm (as discussed briefly at the end of Section V-A). Disjoint paths in general multi-layer graphs resembles work on stable flows [15]. The case of general conflict-free confluent flows uses ideas from this work but significant additional complexity must be managed (Section V-B).

II. LP RELAXATIONS AND MONOTONE NETWORKS

The algorithms of Chen et al. [6] for uncapacitated networks are based upon the following standard network flow relaxation for confluent flows. Recall, we are given a directed graph $G = (V, A)$ and sink node r. Each node v has a demand $d(v)$ of traffic to be routed to the root r, and a *capacity* $\mu(v)$, which is a bound on the total traffic that can be handled by v (including its own traffic if $d(v) > 0$). Then, a *d-flow* is an arc-assignment f such that the net-flow out of each node v is $d(v)$; here $d(r) = 0$. We are interested in d-flows such that the total flow through each v is at most $\mu(v)$. We may thus assume that $d(v) \leq \mu(v)$. For a d-flow f, let $L_f(v)$, or simply $L(v)$, denote the *load* at node v, that is, the total flow out of v. We also denote by $C_f(v)$ the (node) *congestion* $L(v)/\mu(v)$ at v, and the congestion of f is $C_f = \max_v C_f(v)$. The flow is *feasible* if $C_f \leq 1$. We can find a feasible d-flow in polytime by traditional methods. To obtain Theorem I.1, Chen et al. [6] round the d-flow to a confluent flow with a logarithmic blow-up in congestion for capacity minimization (and with only a constant loss in total demand for demand maximization).

A. Bad Examples for Capacitated Networks

Two fundamental difficulties arise when we attempt to apply such an approach to capacitated networks. First, for uncapacitated networks the worst congestion must arise at a neighbour of the sink r. However, for capacitated networks, worst case congestion may occur a node anywhere in the network. This property causes the potential function arguments used in [6] to fail.

Second, an even more critical difficulty is that the standard flow formulation can never give good approximation bounds in capacitated networks. For example, it is not always possible to convert a standard d-flow into a confluent flow with low congestion. To see this, consider the instance in Figure 1. It is easy to see that the minimum node congestion of a (standard) d-flow is 1. But the congestion of any confluent flow is n, since the n units of flow that pass through node x must then pass through an out-neighbour of x of capacity 1. Clearly, this factor n gap also holds for the demand maximization objective.

The polynomial gap in the specific case of Figure 1 can be fixed quite easily. Note that the total load at x in any confluent flow must be at most 1 (since it only sees capacity 1 nodes). Hence we could truncate capacity of x to be 1 without affecting the existence of a confluent flow. After we do this, there is no longer a feasible network flow. We can generalize this truncation technique as follows. Suppose there is a cut-set S that separates v from r in G. Then the maximum amount that can be sent confluently from v to r is at most $\max_{u \in S} \mu(u)$. Consequently, if $\mu(v)$ exceeds this amount then we may truncate it to $\max_{u \in S} \mu(u)$. We say that an arc (i, j) is *monotone* if the capacities satisfy $\mu(i) \leq \mu(j)$. Thus, after this truncation process, for each

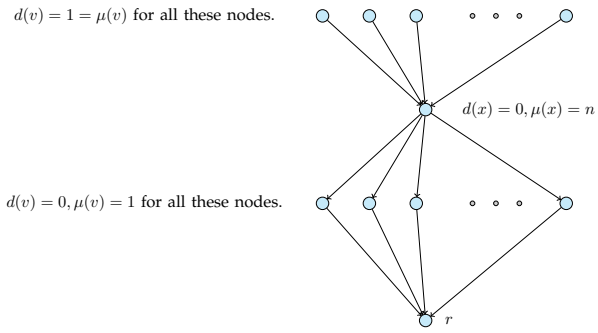

Figure 1. A Bad Example with Unit-Demands.

v, there must be a path from v to r consisting entirely of monotone arcs, that is, a *monotonic path*. Moreover, if the modified instance does not possess a d-flow, then the original instance did not have a confluent flow.

Unfortunately, this truncation procedure may still lead to polynomial gaps. Thus, we must strengthen the procedure further. To understand why this is the case, we now discuss weak and strong network routings.

B. Strong and Weak Routings

We say that a routing on a tree T is *strong* if each demand $d(v)$ routes only through nodes u with $d(v) \leq \mu(u)$. Observe that if we relax the capacity constraint by a factor α, the routings that are not strong (i.e. *weak* routings) may become admissible. If the objective is to simply augment network capacity so that all demands can be routed then weak routings are appropriate On the other hand, if for example the objective is to find a tree T that can route all the demands in some limited number of time periods (rounds), then strong routings are necessary.

Recall that Theorem I.3 [20] states that it is hard to obtain strong routings unless we allow a polynomial blow-up in congestion. Moreover [20] contains an example in which every node has a monotonic path to the root but where a confluent flow requires polynomial more congestion than a d-flow. This arises because a d-flow may use non-monotonic paths, and hence produce weak routings, even if a monotonic path exists.

This motivates us to strengthen our monotonicity constraint further. Clearly, we can avoid weak routings if every path in the network is monotonic. Specifically, we say that G is a *monotonic network* if $\mu(i) \leq \mu(j)$ for every arc (i, j). In a monotonic network, every path from v to r is a monotonic path. So, in the next section, we will focus on monotonic networks and present a polylogarithmic approximation algorithm for (strong) congestion minimization. As described in Section I-B, this algorithm for congestion minimization in monotonic networks will then be used as a subroutine of the polylogarithmic approximation algorithm for demand maximization in non-monotonic networks.

III. Congestion Minimization in Monotonic Networks

In this section, we give the polylogarithmic approximation algorithm for capacity minimization in monotonic networks. The proof is deferred to the full version.

Theorem III.1. *Let $G = (V, A)$ be a monotone network. If there is a feasible d-flow in G, then we can find in polynomial time, a confluent flow with congestion $O(\log^4 n)$, even if the network does not satisfy the* NBA. □

IV. A Multi-Layer Flow Formulation

We now present a new LP formulation for confluent flows based upon a monotonic auxiliary digraph. Let $G = (V, A)$ be a node-capacitated network with root r. Let $k = \lfloor \log(\mu_{max}) \rfloor + 1$. Recall that by our preprocessing we have $\mu_{\max} \leq n$. We then create a *k-layered* auxiliary graph H from G as follows. Set

$V(H) = V(H_0) \cup V(H_1) \cup \ldots V(H_{k-1}) \cup \{r\}$, where $V(H_\ell)$ is a copy of $V(G) \setminus r$. We denote by v^ℓ the copy of $v \in V(G)$ in the ℓth layer H_ℓ. The node v^ℓ will have capacity 2^ℓ. For each arc $(u, v) \in G$, there is a *vertical* arc (u^ℓ, v^ℓ) in H if both u and v have capacity at least 2^ℓ in G. Furthermore, H contains each *horizontal* arc $(v^\ell, v^{\ell+1})$, for each $0 \leq \ell \leq k-2$. Finally, if v is a neighbour of r then we add an arc from (v^{k-1}, r). An example of a 3-layered network is shown in Figure 2. We emphasize the fact that the layered graph H is monotonic.

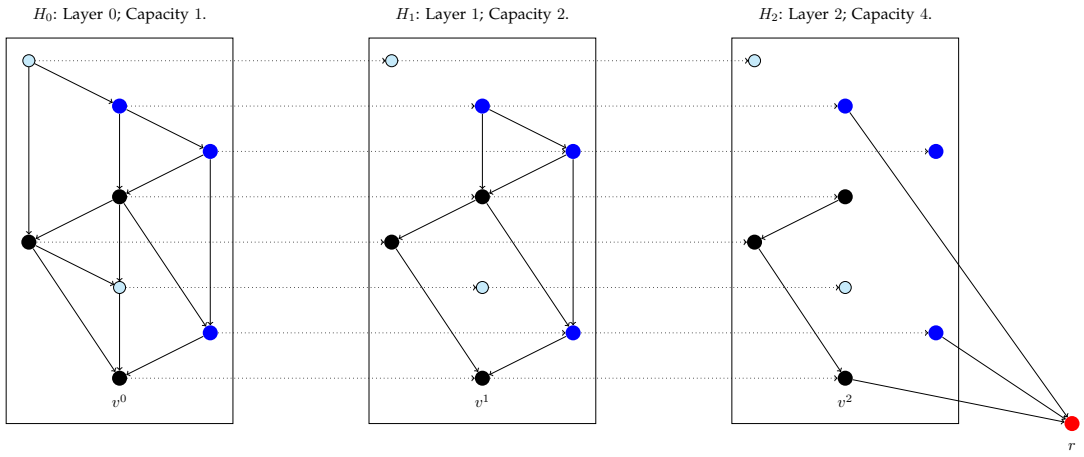

Figure 2. A 3-layer auxiliary network.

Now consider the demands $d(v)$ in some group after our preprocessing step. Recall, after preprocessing, that our instance has unit-demands (and integral capacities). Hence, we can use standard techniques to obtain an integral maximum flow that upper bounds the maximum routable demand of any confluent flows. The bad example in Figure 1, however, shows that such a maximum flow in G may vastly over-estimate the optimal confluent flow. Fortunately, as we shall see, the maximum flow in the multi-layer graph eliminates this problem. Consequently, our starting point is to solve the maximum flow problem in the multi-layer instance, where we place the demand of node v at its copy v^0 in Layer 0. Observe, that this is indeed a valid upper bound as any confluent flow in G can be routed in H. Because H is monotonic, we may then apply Theorem III.1 to give:

Theorem IV.1. *Suppose the layered network H contains a confluent routing of the unit-demands with congestion one. Then we can find, in polynomial-time, a confluent flow in H with congestion $O(\log^4 n)$ that routes all the demands.*

So by Theorem IV.1 we have found a confluent flow h in the layered graph H with polylogarithmic congestion. Unfortunately, since there are a logarithmic number of copies of each node v in H, it may be the case that v has logarithmic out-degree in the original network when we map h back to G. In particular, two copies v^ℓ and $v^{\ell'}$ of v may *conflict* – they have outgoing vertical arcs to different out-neighbours in their respective layers. In the next section we will show how to eradicate conflicts to obtain a true confluent flow in G. First, we remark that Theorem IV.1 allows us to obtain non-trivial congestion guarantees for routing every demand in a capacitated network using a generalization of a confluent flow called a k-furcated flow, provided k is logarithmic. We explain this and provide a proof in the full version.

V. Conflict-Free Routing

So we have found a confluent flow h with $O(\log^4 n)$ congestion in the layered graph H defined in Section IV. This routing has "conflicts" (cf. Section I-B, or the definition below) that we need to resolve.

To do this we will take the support of h in the layered graph and add all the remaining horizontal arcs to form a layered graph $\mathcal{H} \subseteq H \setminus \{r\}$. Our goal is to find a large collection of demands that can be routed confluently in $\mathcal{H}$ without conflicts.

A. Multi-Layer Disjoint Paths

Let's first consider a special case that will help motivate the techniques use in the general case. Assume that, after removing r, the support of h consists of a collection of node-disjoint paths. Let these paths, say $\mathcal{P} = \{P_1, P_2, \ldots, P_\lambda\}$, go from *sources* $S = \{s_1, s_2, \ldots, s_\lambda\} \subseteq V_0$ to *sinks* $T = \{t_1, t_2, \ldots, t_\lambda\} \subseteq V_{k-1}$ in the layered graph $H \setminus \{r\}$. Let $A_v = \{(v^i, v^{i+1}) : 1 \leq i \leq k-1\}$ be the *horizontal path* corresponding to node v. Let $\mathcal{H} = \bigcup_v A_v \cup \bigcup_{j=1}^{\lambda} P_j$. Observe that, because the P_j are disjoint, the graph $\mathcal{H} \subseteq H \setminus \{r\}$ has maximum in-degree and maximum out-degree at most two.

Now recall that $\mathcal{P} = \{P_1, P_2, \ldots, P_\lambda\}$ may not induce a confluent flow in the original network. In particular, we say that P_i, P_j in $\mathcal{H}$ have a *conflict* at $v \in V(G)$ if there exist $p < q$ such that (say) $v^p \in P_i, v^q \in P_j$ and (v^p, v^{p+1}) is not an arc of P_i.

We are interested in finding a collection of *conflict-free* node-disjoint paths as these will induce a confluent flow in the original network. Unfortunately, the existence of a large node-disjoint collection of paths in the multilayer graph does not apriori yield a large node-disjoint routing in the original graph G: multilayer examples can be constructed containing a large collection of paths in H in which every pair of paths has a conflict! Consequently, without some re-routing of the paths we cannot hope to find a large collection for the original graph. Our goal now is to describe an algorithm allowing such a successful rerouting.

In this section, as we are interested in node-disjoint paths in $\mathcal{H}$ we may ignore the node capacities (or, equivalently, assume the capacity of each node v^ℓ is 1 regardless of the layer ℓ.) Furthermore, during the course of the algorithm we will maintain a series of networks

$$\mathcal{H} = \mathcal{H}^0 \supseteq \mathcal{H}^1 \supseteq \mathcal{H}^2 \supseteq \cdots \supseteq \mathcal{H}^\tau.$$

Thus, if there is no path from s_i to T in $\mathcal{H}^t$ then there will be no path from s_i to T in $\mathcal{H}^{t'}$, for any $t' > t$. For any source s_i that has a path to T in $\mathcal{H}^t$, we highlight a unique path as follows. Perform a depth first search in $\mathcal{H}^t$ from s_i until we reach a vertex $T = \{t_1, t_2, \ldots, t_\lambda\}$. During the search we give priority to searching horizontal over vertical arcs. (Since there is at most one horizontal arc and one vertical arc exiting any node, this search is indeed uniquely defined.) If the search succeeds, we call the path to T the *probe path* for s_i, denoted by Q_i^t. We have the following simple observations.

Observation V.1. *If $P_i \subseteq \mathcal{H}^t$, then a probe path Q_j^t exists.* □

Observation V.2. *At any time t, if two probe paths Q_i^t and Q_j^t intersect at v^p then they follow the same path from v^p to T.* □

Observation V.3. *If $Q_i^t \subseteq \mathcal{H}^{t+1}$ then $Q_i^{t+1} = Q_i^t$.*

Furthermore, we will ensure that only vertical arcs may be removed as the network changes from $\mathcal{H}^t$ to $\mathcal{H}^{t+1}$.

Lemma V.1. *At any time t, any two probe paths Q_i^t and Q_j^t are conflict-free.*

Proof. Suppose not. Then there exist $p < q$ such that $v^p \in Q_i^t, v^q \in Q_j^t$ and (v^p, v^{p+1}) is not an edge of Q_i^t. But $\mathcal{H}^t$ contains the horizontal path $\{v^p, v^{p+1}, \ldots, v^q\}$. So, by the priority ordering in the DFS, the probe path Q_i^t from s_i should have taken this path and then merged with Q_j^t. Thus the probe paths are conflict-free. □

Let's now describe the algorithm. At time t, we will partition the set S of source nodes into three sets: *active*, *unprocessed* and *discarded*. We denote these by $\mathcal{A}^t, \mathcal{U}^t$ and $\mathcal{D}^t$, respectively. Associated with the set of active sources there will be a collection of paths $\mathcal{P}^t$. Every active source node $s_i \in \mathcal{A}^t$, except at most one,

will be *connected*; that is, there is a path $P_i^t \in \mathcal{P}^t$ from s_i to T. (The path P_i^t will be the probe path Q_i^t in $\mathcal{H}^t$ from s_i to T.) The remaining active node (if it exists) will be called *disconnected*.

If there is no probe path Q_j^t in $\mathcal{H}^t$ from s_j to T then we include s_j in the set $\mathcal{D}^t$ of discarded nodes. Clearly $\mathcal{D}^t \subseteq \mathcal{D}^{t+1}$ as $\mathcal{H}^t \supseteq \mathcal{H}^{t+1}$. Source nodes that are neither active nor discarded are unprocessed.

At time $t = 0$, every source node is inactive $\mathcal{U}^0 = S$ and $\mathcal{P}^0 = \emptyset$. At the start of time t, we already have a network $\mathcal{H}^t$ and a collection of paths $\mathcal{P}^{t-1}$ (where $P_j^{t-1} = Q_j^{t-1}$ is a probe path from s_j to T in $\mathcal{H}^{t-1}$). If any active node $s_i \in \mathcal{A}^t$ is disconnected then we attempt to grow a probe path $Q_i^t \in \mathcal{H}^t$ from s_i to T. If every active node $s_i \in \mathcal{A}^t$ becomes connected, then we select an arbitrary unprocessed source $s_i \in \mathcal{U}^t$ and attempt to grow a probe path from it.

Let's consider the outcome of growing a probe path for a disconnected source. If we do not succeed to find a probe path for s_i, then it is discarded. Namely, set $\mathcal{D}^t := \mathcal{D}^{t-1} \cup \{s_i\}$; $\mathcal{P}^t := \mathcal{P}^{t-1}$. Otherwise the probe path Q_i^t exists and we make s_i active (if it wasn't already). Now, if Q_i^t is node-disjoint from each probe path in $\mathcal{P}^{t-1}$ then we set $\mathcal{P}^t := \mathcal{P}^{t-1} \cup Q_i^t$. Otherwise, the probe path Q_i^t exists but it meets and merges at a node v^p with some probe path Q_j^{t-1} in $\mathcal{P}^{t-1}$. In this case, we will ensure that Q_i^t must have entered v^p along the horizontal arc (v^{p-1}, v^p) whilst Q_j^{t-1} entered via the vertical arc in layer p. We then set $\mathcal{P}^t := (\mathcal{P}^{t-1} \setminus Q_j^{t-1}) \cup Q_i^t$. Thus s_i is connected but s_j is now disconnected. We must now update $\mathcal{H}^t$. Let $w(Q_i^t)$ be the set of vertical arcs that are not in Q_i^t but that enter some node of Q_i^t. We call $w(Q_i^t)$ the *whiskers* of Q_i^t. We set $\mathcal{H}^{t+1} := \mathcal{H}^t \setminus w(Q_i^t)$. We emphasize that $\mathcal{H}^{t+1}$ does not contain the whisker of any path P that was selected during any earlier time period – thus whiskers cannot be added back into $\mathcal{H}^{t+1}$ even if P is no longer in the collection $\mathcal{P}^t$. Note this is also why a new probe path Q_i^t must use the horizontal edge when it merges with an existing probe path Q_j^{t-1}. We repeat this process until the time τ when every node is either connected or discarded.

Observe that Lemma V.1 implies that the probe paths in $\mathcal{P}^\tau$ form a confluent flow (in fact node-disjoint flow) in the original network. We now show that the set of output paths $\mathcal{P}^\tau$ has large cardinality.

Theorem V.2. *The set $\mathcal{P}^\tau$ has cardinality at least $\frac{1}{k} \cdot \lambda$.*

Proof. At termination, every node is either active and connected or discarded. By construction, a discarded source s_i has no probe path in $\mathcal{H}^\tau$. In particular, by Observation V.1, at least one arc in the original path P_i must be missing from $\mathcal{H}^\tau$. We want to show that at least one of the missing arcs of P_i is the whisker of an output path P_j^τ. So take the arc $e = (u^p, v^p)$ furthest along P_i that is absent from $\mathcal{H}^\tau$. The arc e was removed at time $t \leq \tau$ because there was a probe path $Q_{j_1}^t$ that used the arc $f = (v^{i-1}, v^i)$. If at time τ, some path $P_{j_2}^\tau$ includes f then e is a whisker of an output path as desired. So let $P_{j_3}^{t'}$ from s_{j_3} be the last path selected that used f, where $t \leq t' \leq \tau$. When $P_{j_3}^{t'}$ was replaced, no probe path from s_{j_3} could go through f and then reach a sink. Otherwise such a path would have been chosen at time $t' + 1$. In particular the subpath of P_i from v^p to T must have been missing an arc e'. But e' is further along P_j than e, a contradiction. Thus e is a whisker of an output path.

Now, by construction of the network, each output path can intersect at most one path from $\mathcal{P} = \{P_1, P_2, \ldots, P_\lambda\}$ in any layer. Thus each output path has at most $k - 1$ whiskers (since there is none in layer 0) and hence is charged by at most $k - 1$ paths in $\mathcal{P}$. The theorem follows. □

A connection to stable matchings can be seen intuitively as follows. One defines a bipartite graph with bipartition $A \cup B$ where A consists of the set of sources in the 2-layer graph, and B the set of sinks. One includes an edge uv for each path from u to v in the 2-layer graph. One may then define precedence orders for a node $u \in A$ to prefer edges (i.e., paths in the 2-layer graph) which use a horizontal edge earlier, when the path is traversed from u. Similarly, one may consider a precedence order (somewhat opposite to the ones for A) for each $v \in B$, where the node v prefers paths which use a horizontal edge as early as possible when traversed from the node v. One checks that a conflict-free collection of paths corresponds to a stable matching in this stable matching instance.

B. Multi-layer Confluent Flows

We now consider the general case. Again we have a collection of paths $\mathcal{P} = \{P_1, P_2, \ldots, P_\lambda\}$ from S to T in the layered graph. This time the paths need not be disjoint but instead form a confluent flow in the routing network $\mathcal{H} = \mathcal{H}_0$. Again, each node in $\mathcal{H}$ has at most one outgoing vertical arc and at most one outgoing horizontal arc. Consequently, we can grow probe paths in each $\mathcal{H}^t$ according to the same priority rule as before.

On the other hand, we have two differences from the disjoint-paths case. First, the graph $\mathcal{H}$ is now capacitated; a node v^q in layer q has capacity 2^q. (We may assume, by rerounding, that this capacity already includes the $O(\log^4)$ congestion factor). Second, a vertex v^q may have in-degree greater than two, as the confluent flow induces in-arborescences. Our goal is to find a collection of conflict-free paths $\mathcal{P}^\tau = \{P_1^\tau, P_2^\tau, \ldots, P_\lambda^\tau\}$ (where some of the P_i^τ may be empty) that contains at least $\Omega(\frac{1}{k} \cdot |\mathcal{P}|)$ non-empty paths. We prove how to do this in the full version by modifying the probe-path algorithm to take into account capacities and large in-degrees.

Our main results, Theorems I.5 and I.7 then follow as described in Section I-B.

VI. Acknowledgements

We are grateful to two reviewers of the extended conference submission. Many of their excellent suggestions were adopted and the remainder will be adopted in the full version of the paper.

References

[1] R. Bhatia, N. Immorlica, T. Kimbrel, V. Mirrokni, S. Naor, and B. Schieber. Traffic engineering of management flows by link augmentations on confluent trees. *Theory of Computing Systems*, 42(1):2–26, 2008.

[2] P. Bonsma, J. Schulz, and A. Wiese. A constant factor approximation algorithm for unsplittable flow on paths. In *Proceedings of the Fifty-Second Symposium on Foundations of Computer Science (FOCS)*, pages 47–56. IEEE, 2011.

[3] C. Chekuri and J. Chuzhoy. Large-treewidth graph decompositions and applications. In *Proceedings of the Forty-Fifth ACM Symposium on Theory of Computing (STOC)*, pages 291–300, 2013.

[4] C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *Proceedings of the Twelveth Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 42–55. Springer, 2009.

[5] C. Chekuri, M. Mydlarz, and B. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3(3):27, 2007.

[6] J. Chen, R. Kleinberg, L. Lovasz, R. Rajaraman, R. Sundaram, and A. Vetta. (Almost) tight bounds and existence theorems for confluent flows. *Journal of the ACM*, 54(4), #16, 2007.

[7] J. Chen, R. Rajaraman, and R. Sundaram. Meet and merge: Approximation algorithms for confluent flows. In *Proceedings of the Thirty-Fifth ACM Symposium on Theory of Computing (STOC)*, pages 373–382, 2003.

[8] J. Chen, R. Sundaram, M. Marathe, and R. Rajaraman. The confluent capacity of the Internet: congestion vs. dilation. In *Proceedings of the Twenty-Sixth IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[9] J. Chuzhoy. Routing in undirected graphs with constant congestion. In *Proceedings of the Forty-Fourth ACM Symposium on Theory of Computing (STOC)*, pages 855–874, 2012.

[10] J. Chuzhoy and S. Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. In *Proceedings of the Fifty-Third Symposium on Foundations of Computer Science (FOCS)*, pages 233–242, 2012.

[11] Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19(1):17–41, 1999.

[12] P. Donovan, B. Shepherd, A. Vetta, and G. Wilfong. Degree-constrained network flows. In *Proceedings of the Thirty-Ninth ACM Symposium on Theory of Computing (STOC)*, pages 681–688, 2007.

[13] D. Dressler and M. Strehler. Capacitated confluent flows: complexity and algorithms. In *Proceedings of the Seventh International Conference on Algorithms and Complexity (CIAC)*, pages 347–358, 2010.

[14] D. Dressler and M. Strehler. Polynomial-time algorithms for special cases of the maximum confluent flow problem. *Discrete Applied Mathematics*, 163:142–154, 2014.

[15] T. Fleiner. On stable matchings and flows. In *Graph Theoretic Concepts in Computer Science*, pages 51–62. Springer, 2010.

[16] J. Kleinberg. Single-source unsplittable flow. In *Proceedings of the Thirty-Seventh Symposium on Foundations of Computer Science (FOCS)*, pages 68–77, 1996.

[17] L Lovász. A homology theory for spanning tress of a graph. *Acta Mathematica Hungarica*, 30(3):241–251, 1977.

[18] L. Seguin-Charbonneau and F.B. Shepherd. Maximum edge-disjoint paths in planar graphs with congestion 2. In *Proceedings of the Fifty-Second Symposium on Foundations of Computer Science (FOCS)*, pages 200–209, 2011.

[19] B. Shepherd and G. Wilfong. Multilateral transport games. In *Proceedings of the International Network Optimization Conference (INOC)*, volume B2, pages 378–375, 2005.

[20] F. Bruce Shepherd and Adrian Vetta. The inapproximability of maximum single-sink unsplittable, priority and confluent flow problems. *arXiv*, 2015.