A Primal-Dual Algorithm for Multicommodity Flows and Multicuts in Treewidth-2 Graphs

Tobias Friedrich
Hasso Plattner Institute, Universität Potsdam, Germany

Davis Issac
Hasso Plattner Institute, Universität Potsdam, Germany

Nikhil Kumar
Hasso Plattner Institute, Universität Potsdam, Germany

Nadym Mallek
Hasso Plattner Institute, Universität Potsdam, Germany

Ziena Zeif
Hasso Plattner Institute, Universität Potsdam, Germany

Abstract

We study the problem of multicommodity flow and multicut in treewidth-2 graphs and prove bounds on the multiflow-multicut gap. In particular, we give a primal-dual algorithm for computing multicommodity flow and multicut in treewidth-2 graphs and prove the following approximate max-flow min-cut theorem: given a treewidth-2 graph, there exists a multicommodity flow of value f with congestion 4, and a multicut of capacity c such that $c \leq 20f$. This implies a multiflow-multicut gap of 80 and improves upon the previous best known bounds for such graphs. Our algorithm runs in polynomial time when all the edges have capacity one. Our algorithm is completely combinatorial and builds upon the primal-dual algorithm of Garg, Vazirani and Yannakakis for multicut in trees and the augmenting paths framework of Ford and Fulkerson.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Approximation Algorithms, Multicommodity Flow, Multicut

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2022.55

Category APPROX

Funding This research was partially funded by the HPI Research School on Data Science and Engineering.
Ziena Zeif: HPI Research School.

1 Introduction

Given an undirected graph with edge capacities and k source-sink pairs, the *maximum multicommodity flow problem* asks for the maximum amount of flow that can be routed between the source-sink pairs. If the flows are restricted to be integral, then the problem is called the *maximum integral multicommodity flow*. An important special case of this problem is the *maximum edge disjoint paths problem*, where the objective is to find the maximum number of source-sink pairs that can simultaneously be connected by edge-disjoint paths. In a multicommodity flow with *congestion* c, an edge may be used by up to c flow paths. The maximum edge disjoint paths problem is NP-Hard, even in very restricted settings such as when the graph is series-parallel [14]. Maximum edge disjoint paths problem is hard to approximate in general (even with congestion, see Section 2.1 for further discussion). Multicommodity flow problems have been studied extensively over the last five decades and find extensive applications in VLSI design, routing and wavelength assignment etc. [17].

A natural dual to the maximum multicommodity flow problem is the *minimum multicut problem*. Given an edge-capacitated graph with k source-sink pairs, a multicut is a set of edges whose removal disconnects all the source-sink pairs, and the capacity (or value) of the cut is the sum of capacities of the edges in it. The value of any feasible multicommodity flow is at most the capacity of any feasible multicut. The ratio of the values of the minimum multicut and maximum multicommodity flow is called the *multiflow-multicut gap*. The ratio of the values of the minimum multicut and maximum multicommodity flow with congestion-c is called the *multiflow-multicut gap* with congestion c. In case c is 1 or 2, we call it the integral or half-integral multiflow-multicut gap respectively. Minimum multicut is NP-Hard to compute, even in very restricted setting such as trees [11]. More precisely, it is known to be equivalent to the vertex cover problem in stars with unit weights [11], which implies that it is APX-Hard in series-parallel graphs. There is a rich literature on proving bounds on the multiflow-multicut gap. Perhaps the most famous of them is the max-flow min-cut theorem of Ford and Fulkerson [7], which states that the value of the minimum multicut is equal to the maximum (integral) flow when $k = 1$. Hu [12] extended the result of Ford and Fulkerson to show that the multiflow-multicut gap is 1 even when $k = 2$. Another tight example, closely related to our work, is the case where the graph obtained by adding an edge for each source-sink pair is series-parallel [5]. There are many other special cases where the multiflow-multicut gap is 1, for example when G is a path or a cycle, but in general it can be arbitrarily large. Garg et al. [10] proved a tight bound of $\Theta(\log k)$ on the multiflow-multicut gap for any graph G. For K_r minor-free graphs, Tardos and Vazirani [16] used the decomposition theorem of Klein et al. [13] to prove a bound of $O(r^3)$ on the multiflow-multicut gap. The integral multiflow-multicut gap can be $\Omega(\sqrt{|V|})$, even for planar graphs (see Figure 1).

Garg et al. [11] gave a tight bound of 2 on the integral multiflow-multicut gap when G is a tree. For graphs of treewidth r, Abraham et al. [1] gave a bound of $O(r)$ on the multiflow-multicut gap by rounding a natural linear programming relaxation. Chekuri et al. [3] and Ene et al. [6] showed how to round a fractional multicommodity flow solution into an integral one by losing a factor of $O(r^3)$. Combining their results gives a bound of $O(r^4)$ on the integral multiflow-multicut gap for graphs of treewidth r. Note that this implies a $O(1)$ bound on the multiflow-multicut gap for treewidth 2 graphs. All the results mentioned above are algorithmic in nature and also imply an approximation algorithm for the (integral) multicommodity flow and multicut problems. Except for the case when G is a tree, all the results mentioned above are proved by rounding a natural linear programming relaxation to the problem.

We extend the augmenting paths framework of Ford and Fulkerson [7] to develop a primal-dual algorithm for multiflow and multicut for treewidth 2 graphs (see Theorem 2). It is a well known fact that the augmenting paths framework cannot be used for multicommodity flows in general. To the best of our knowledge, this is the first time augmenting paths framework has been adapted (in a non-trivial manner) for developing an algorithm for multicommodity flows and multicuts.

A simple topological obstruction of Garg et al. [11] shows that the integral multiflow-multicut gap is $\Omega(r)$ for graphs with treewidth r (see Figure 1). Chekuri et al. [2] and Ene et al. [6] raised the question if the integrality gap of the natural linear programming for multicommodity flows is $O(r)$ for graphs with treewidth r. We believe that the topological obstruction of Garg et al. [11] gives the best possible lower bound on the integral multiflow-multicut gap for graphs of treewidth r. To this end, we make the following conjecture, which strengthens the one stated by Ene et al. [6].

▶ **Conjecture 1.** *The integral multiflow-multicut gap for graphs with treewidth r is $\Theta(r)$.*

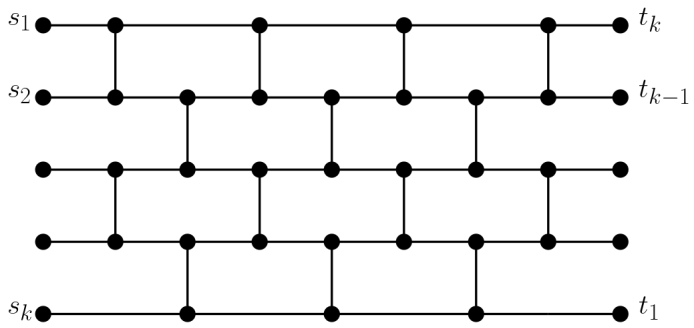


Figure 1 In the above instance, all the edges have unit capacity and hence only one source-sink pair can be connected by edge-disjoint paths. We need at least k edges to disconnect all the source-sink pairs and hence the integral multiflow-multicut gap is at least $\Omega(k)$. The graph has a treewidth of $\Theta(k)$. This shows that the integral multiflow-multicut gap can be $\Omega(r)$ for graphs with treewidth r.

It is known that the integrality gap for the linear programming relaxation for the multicut and the integer multicommodity flow for treewidth r graphs is $\Omega(\log r)$ and $\Omega(r)$ respectively. Hence, any algorithm which rounds the linear programming relaxation for multicommodity flow and multicut separately won't be able to resolve this conjecture. We believe that a primal-dual algorithm, which works with multicommodity flow and multicut simultaneously will lead to the resolution of this conjecture. We also believe that the techniques we develop in this paper makes important progress towards developing such an algorithm.

2 Our Contribution

As already noted in Section 1, results of Abraham et al. [1] and Ene et al. [6] imply an $O(1)$ bound on the (integral) multiflow-multicut gap for treewidth 2 graphs, albeit with a large (unspecified) constant. Our main technical contribution is developing the first primal-dual algorithm for multiflow and multicut for treewidth 2 graphs. We prove the following approximate max-flow min-cut theorem for treewidth 2 graphs (see Section 3 for precise definitions):

▶ **Theorem 2.** *Let G be an undirected, (integer) edge capacitated treewidth 2 graph and $\{(s_i, t_i)\}_{i=1}^{k}$ be the source-sink pairs. Then there exists an integral multicommodity flow of value f with congestion 4 and a multicut of value c such that $c \leq 20f$. Furthermore, there exists a primal-dual algorithm that computes such a flow and cut in time polynomial in size of the graph and the largest capacity. For unit capacity graphs, the algorithm runs in polynomial time.*

Our proof of Theorem 2 is completely combinatorial and does not require us to solve a linear program. It is based on the primal-dual framework. This leads to a more explicit algorithm and sheds further light on the structure of the multicuts and multicommodity flows in treewidth 2 graphs. All previous algorithms for computing multicommodity flows and multicuts were based on rounding the standard linear programming relaxation (except for some special cases, see Section 2.1). In many combinatorial optimization problems, algorithms based on the primal-dual schema give (near) optimal bounds on the approximation ratio, and we hope that further extensions of our approach will lead to tight results in the context of this problem as well. We would also like to point out that the bounds of Theorem 2 are the best known.

The broad outline of our proof follows the Ford-Fulkerson algorithm for computing the maximumum (s,t)-flow and minimum (s,t)-cut in a graph. Since multiflows and multicuts are linear programming dual of each other, our algorithm can also be seen as a primal-dual algorithm. In each iteration, we increase the total flow by performing an augmenting step, ie. rerouting previously routed flow paths. This is done by generalizing the well known augmenting paths framework of Ford and Fulkerson [7] for single commodity flow. This generalization requires new ideas as it is well known that the augmenting paths framework can not be used directly for multicommodity flows. We then use the reachability graphs defined by the flows at the end of the algorithm to find a multicut for the instance, which can also be seen as generalisation of the cut-picking algorithm of Ford-Fulkerson [7].

The problem of computing minimum multicut can be formulated as an integer linear program. We can relax the integrality constraints to obtain a linear programming (LP) relaxation for multicut. The ratio between the optimum solution to the integer program and the LP relaxation is called the integrality gap of the relaxation. Theorem 2 also implies the same bound on the integrality gap of the integer programming relaxation for multicut in treewidth 2 graphs.

In Section 3, we formally define the problem statement and state the connection between treewidth 2 and series-parallel graphs. In Section 4, we give a quick overview of the augmenting paths algorithm of Ford-Fulkerson [7] for the single commodity case. In Section 5, we illustrate the basic ideas of our algorithm for a special case, ie. parallel-path graphs. In Section 7 and Section 8, we give the full algorithm for series-parallel graphs. We then go on to show how to pick a multicut in Section 9.

2.1 Other Related Work

Garg et al. [11] gave a primal-dual 2-approximation algorithm for finding an integral multicommodity flow and multicut for trees. Their result also implies a tight bound on the integral multiflow-multicut gap for trees. By combining the results of [8, 9], we can obtain a primal-dual algorithm for computing a multicut and integral flow when the graph obtained by adding an edge for every source-sink pair to G is planar. These also imply a tight half-integral multiflow-multicut gap of 2 and integral multiflow-multicut gap of 4 for such instances. To the best of our knowledge, there are no other completely combinatorial algorithm proving bounds on the multiflow-multicut gap for non-trivial class of instances.

The problem of finding maximum edge disjoint paths is NP-Hard, even in very restricted settings [14]. There is an $O(\sqrt{n})$ approximation algorithm for finding maximum edge disjoint paths in general (undirected) graphs on n vertices [2]. This also matches the integrality gap of the natural linear programming relaxation for the problem [11]. Recently, Chuzhoy et al. showed that it is not possible to approximate the maximum edge disjoint paths problem better than $2^{\Omega(\log^{1-\epsilon} n)}$ under reasonable hardness assumptions and it is an outstanding open problem to improve the $O(\sqrt{n})$ approximation algorithm, even for planar graphs. If we relax the edge-disjointedness condition and allow every edge to be used by up to c paths for some integer $c \geq 2$, then the problem is called the maximum edge disjoint paths with congestion c. A long line of impressive work culminated in a $O(\text{polylog } n)$ approximation algorithm for general graphs [4] and a constant factor approximation algorithm for planar graphs [15] when a congestion of 2 is allowed. Both these results also imply the same bound on the integrality gap of the natural linear programming relaxation. The exact integrality gap of the maximum edge disjoint paths with congestion 2 for K_r minor-free graphs is still not known and is an interesting open question.

3 Preliminaries

Let $G = (V, E)$ be a simple undirected graph with edge capacities $c\colon E \to \mathbb{Z}_{\geq 0}$; we call this the supply graph. Let $H = (V, F)$ be a simple graph each edge of which corresponds to a commodity and the endpoints of that edge are the source-sink of that commodity. H is the demand graph and its edges the demands.

Let $\mathcal{P}$ be the set of all paths in G between a source and its corresponding sink. For a path $P \in \mathcal{P}$, we refer to f_P as the value of flow on P. A multiflow $f\colon \mathcal{P} \to \mathbb{R}_{\geq 0}$ is feasible if for every edge $e \in E$, the total flow on all paths containing the edge, $\sum_{P:e\in\mathcal{P}} f_P$, is at most the capacity of the edge, $c(e)$. We say that a multiflow has congestion l if the flow paths are allowed to use an edge up to l times its capacity, ie. $\sum_{P:e\in\mathcal{P}} f_P \leq l \cdot c(e)$. If the value of flow on every path is an integer (resp. half-integer), then the flow is called an integral (resp. half-integral) multiflow.

A maximum multiflow is a feasible flow f which maximises $\sum_{P\in\mathcal{P}} f_P$. A multicut is a set of edges $E' \subseteq E$ such that every $P \in \mathcal{P}$ contains at least one edge in E'. Equivalently, a multicut is a set of edges whose removal disconnects every source-sink pair. Since a multicut contains an edge of every path in $\mathcal{P}$, the value of any feasible multicut is at least the value of any feasible multiflow. The ratio of the minimum multicut to the maximum (integral/half-integral) multiflow is called the (integeral/half-integral) multiflow-multicut gap.

A cut $S \subseteq V$ is a partition of the vertex set $(S, V \setminus S)$. Let $\delta_E(S)$ denote the edges in E with exactly one endpoint in S. For a subset $E' \subseteq E$ let $c(E')$ be the total capacity of edges in E'. Let $\delta_{\mathtt{min}}(u, v, G)$ denote the minimum value cut between u and v in G.

Series-Parallel Graphs. We will mostly focus on 2-terminal series-parallel graphs as the problem in treewidth-2 graphs can be easily converted to one in 2-terminal series-parallel graphs (see Proposition 3). From now on, we omit 2-terminal series-parallel graphs as simply series-parallel graphs. We will use a well known recursive definition of series-parallel graphs. A series-parallel graph has two distinguished vertices (also called the **merge vertices**) u, v. An edge is a series-parallel graph with its endpoints as the two merge vertices. Starting from an edge, any series-parallel graph can be constructed by two operations: parallel and series composition. Given two series-parallel graphs G_1, G_2 with merge vertices $(u_1, v_1), (u_2, v_2)$, a parallel composition G_p of G_1, G_2 is constructed by setting $u = u_1 = u_2$, $v = v_1 = v_2$ and (u, v) as the merge vertices. Given two series-parallel graphs G_1, G_2 with merge vertices $(u_1, v_1), (u_2, v_2)$, a series composition G_s of G_1, G_2 is constructed by setting $v_1 = u_2$ and (u_1, v_2) as the merge vertices. See Fig. 6 for an illustration. Consider $k \geq 2$ simple node disjoint paths $P_1, P_2, \ldots, P_k$ between two vertices u, v. We call such a graph a *parallel-path graph*. In other words, parallel-path graphs have two distinguished vertices u and v and consist of internally vertex-disjoint u-v paths.

Series-Parallel Tree Decomposition. For a series-parallel graph G, we associate with it a tree-decomposition $T(G)$. This is the canonical tree-decomposition of a series-parallel graph and consists of either 2 or 3 vertices in each bag. The tree-decomposition $T(G)$ can be defined recursively as follows: if G is just an edge $\{u, v\}$ then $T(G)$ consists of a single bag $\{u, v\}$; if G is a parallel-composition of $G_1, G_2, \ldots, G_r$ with merge vertices u and v, then $T(G)$ is obtained by taking the bag $R = \{u, v\}$ as the root and adding edges from R to the root of each of $T(G_1), T(G_2), \ldots, T(G_r)$; if G is a series composition of G_1, G_2 with merge vertices u, v and the common merge vertex of G_1 and G_2 being w, then $T(G)$ is obtained by taking the bag $R = \{u, v, w\}$ as the root and adding edges from R to the root of each of

$T(G_1)$ and $T(G_2)$. We will use T throughout to denote the series-parallel tree-decomposition of the input series-parallel graph G, and for a node X of T, we use T_X to denote the sub-tree of T rooted at X. Also, we use G_X to denote the graph induced in G by the union of vertices in all the nodes in T_X.

We will work with series-parallel graphs in the paper. But the results apply also to treewidth-2 graphs because of the following proposition.

▶ **Proposition 3.** *Given an edge-capacitated treewidth-2 graph G and source-sink pairs T, one can in polynomial time find a series-parallel graph $H \supseteq G$ such that any multicommodity flow with congestion g in H with respect to T is a multicommodity flow with congestion g in G with respect to T, and any multicut of H with respect to T is a multicut of G with respect to T having the same capacity.*

Proof. It is well known that every treewidth-2 graph is the sub-graph of a (2-terminal) series-parallel graph and such a super-graph that is (2-terminal) series-parallel can be found in polynomial time. We add the extra edges to make the graph series-parallel and set their capacities to 0. It is easy to see that then the proposition follows. ◀

For the sake of presentation, we make the following simplifying assumption. Let $v \in V$ be a vertex and suppose that k source-sink pairs are incident on v. Then we add k edges $(v, v_1), (v, v_2), \ldots, (v, v_k)$ to G and set the capacity of each (v, v_i) to be equal to a large number, say $\sum_{e \in E} c_e$. If a source-sink pair (v, t) is incident on v, we replace it by (v_i, t), such that each v_i has exactly one source-sink pair incident on it. We repeat this process for each vertex in the graph and let U be the set of new vertices introduced by this operation. Now every source-sink pair is incident on vertices in U and any vertex has at most one source-sink pair incident on it. Furthermore, there is one to one correspondence between any feasible multiflow and multicut with value at most $\sum_{e \in E} c_e$ in the original and the modified graph. Hence, from now on we assume that exactly one source-sink edge is incident on any vertex of G.

4 Ford-Fulkerson Algorithm for Single Source

We heavily use the augmenting paths framework of Ford-Fulkerson [7] to design our algorithm. We give a brief overview of their algorithm here. Given a source vertex s and a set of sink vertices $T = \{t_1, t_2, \ldots, t_m\}$, we wish to find the maximum amount of flow that can be routed from s to vertices in T. It is convenient to work with a directed network $N = (V, E')$, where each edge $(u, v) \in E$ is replaced by two directed edges (arcs) (u, v) and (v, u) in N. The capacity of each of the arcs is equal to the capacity of the corresponding original edge. All the flow paths are directed from s to T in N. One can show that if a flow of value f can be routed in N, then a flow of value f can be routed in G as well. This allows us to work with N instead of G.

Let F be a set of flow paths directed from s to T in N and $f(e)$ be the flow through arc e in F. We define the residual network with respect to F, $N_F = (V, E')$, as follows: if $f(u, v) \geq f(v, u)$, then we set the capacity of (u, v) to $c_{uv} - f(u, v) + f(v, u)$ and the capacity of the arc (v, u) to $c_{uv} + f(u, v) - f(v, u)$ in N_f. Note that when f is empty, then the capacity of the forward and the backward arcs is equal to the capacity of the original edge in G.

The algorithm works in iterations. In each iteration, we increase the amount of flow from s to T by 1. At the beginning of each iteration, we find the set of reachable vertices R^F in the residual network N^F with respect to the current flow F. If there exists a $t_i \in R_F$, then we augment a unit of flow along a path from s to t_i in N_F. We update our residual graph as

described above and repeat the procedure until some vertex of T is reachable from s in the residual graph. We stop when none of the vertices of T are reachable from s in the residual graph. If the algorithm terminates after f iterations, then there exist f flow paths in the original graph G from s to T. In fact such flow paths can be computed directly from the final residual graph in polynomial time.

Let S be the set of reachable vertices in the residual network at the termination of the algorithm and f be the total number of flow paths routed. Ford-Fulkerson [7] showed that $\delta(S) = f$, ie. the maximum amount of flow from s to T in G is equal to the minimum (total) capacity set of edges which disconnect s from T. This is also known as the max-flow min-cut theorem for sinlge-commodity flow.

Residual Graph for Multicommodity Flow. We can analogously define a residual network N_F of a graph G with respect to any (directed) flow F, and not just the single commodity flow. From now on, we will refer to N_F as the residual network of G with a current (directed) flow F. We will use $f^-(v)$ and $f^+(v)$ to denote the net incoming and outgoing flow incident at the vertex v.

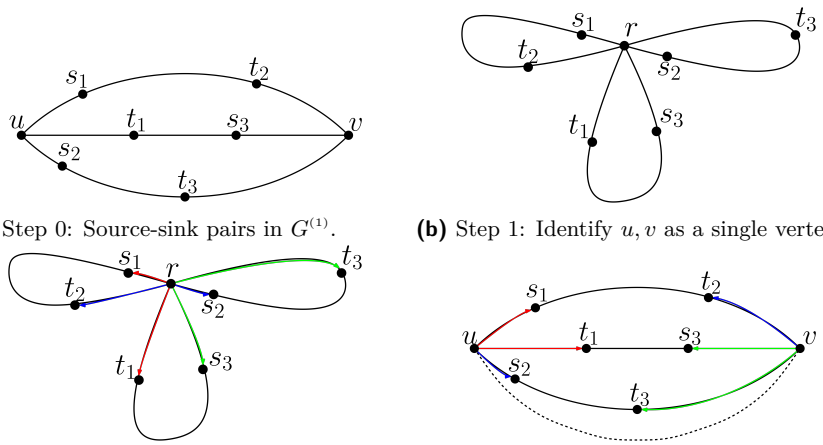
5 Algorithm for Parallel-Path Graphs

To illustrate the basic ideas of our approach, we first describe the algorithm for parallel-path graphs. Let G be a parallel-path graph and (u, v) be its merge vertices. We make a further simplifying assumption that all the source-sink pairs lie on different paths of G. This implies that all the source-sink paths contain either u or v. Let p be the maximum amount of flow that can routed between u and v in G.

Our algorithm works with four copies of G, i.e. $G^{(1)} = G^{(2)} = G^{(3)} = G^{(4)} = G$ each with the same capacities as G. Our flow paths at the end will lie in the union of the four copies. The capacity constraints on the edges will be satisfied within each copy. Thus, we will have a flow with congestion at most 4. We use the augmenting paths framework of Ford and Fulkerson to route flow in $G^{(1)}$. As it is well known, the augmenting paths framework can lead to infeasible flows when applied to a multicommodity setting. We carefully use the edges in $G^{(2)}, G^{(3)}, G^{(4)}$ to *correct* the infeasible flows routed in $G^{(1)}$.

In $G^{(1)}$, we identify u, v as a single vertex and use the Ford-Fulkerson algorithm to construct a flow and cut as follows: let r be the vertex formed by identifying u, v. Observe that r is a cut-vertex and all the source-sink paths go through r. We think of a path between an $s_i - t_i$ pair as the union of two (directed) paths: one from r to s_i and the other from r to t_i. To send f units of flow between an $s_i - t_i$ pair, we first send a flow of value f from r to s_i and then another flow of value f from r to t_i. We call each of these as a **half-flow-path** of the flow between s_i and t_i. Note that all the half-flow-paths are directed away from r. Since every flow path is rooted at r, we treat it as the common source and use the augmenting paths algorithm of Ford-Fulkerson (see Figure 2). We use this process iteratively to route more flow between the source-sink pairs and distinguish between two cases:

Case 1. Suppose the algorithm terminates with a total flow of $f < p$ (recall that p is the maximum u-v flow). Let S be the set of all the reachable vertices from r at the end of the algorithm. If there exists an i such that $s_i, t_i \in S$, then we would have been able to send more flow from r to s_i and r to t_i. Note that an $r - s_i$ path does not overlap with an $r - t_i$ path since s_i and t_i are assumed to be in different paths of the parallel-path graph G.



(a) Step 0: Source-sink pairs in $G^{(1)}$.

(b) Step 1: Identify u, v as a single vertex r.

(c) Step 2: Augment flows from r.

(d) Step 3: Separate u, v and use a (u, v) path in $G^{(2)}$ to get a feasible flow.

Figure 2 Routing flow in a parallel-path graph.

Since it is not possible to send any additional flow between the source-sink pairs, it must be true that S does not contain at least one of s_i, t_i, and hence the edges $\delta(S)$ form a feasible multicut for this instance. Since r was formed by combining u, v, we may not have a feasible flow of value f, i.e. for a source-sink pair, one half-flow-path may be routed from u while the other one is routed from v. To convert this into a feasible flow, we use (at most) f units of u-v flow in $G^{(2)}$. This results in a flow with congestion 2. Since every half-flow path uses at most one edge of the cut $\delta(S)$, we have that the value of the multicut is at most 2 times the total flow routed (with congestion 2).

Case 2. Suppose at some point in the algorithm, that the total flow routed (in $G^{(1)}$) becomes exactly p. Since the maximum u-v flow in G is p, there exists a set of edges, say C, of value p whose removal separates u and v in $G^{(1)}$ (by the max flow-min cut theorem [7]). In this case, we pick a set of cut edges C with total value p in $G^{(1)}$. Let $G_1^{(1)}, G_2^{(1)}$ be the graphs formed after removing the edges in C and let $u \in G_1^{(1)}, v \in G_2^{(1)}$.

Now, let us re-split r into u and v as it was. Each of the half-flow-paths are now rooted at either u or v. If both $G_1^{(1)}$, and $G_2^{(1)}$ do not contain any source-sink pairs within them, we terminate. If there are source-sink pairs that are not separated by the removal of C, we augment flow from u in $G_1^{(1)}$ and from v in $G_2^{(1)}$ to increase our total flow. To do this, we use the augmenting paths algorithm with u (resp. v) as the single source for $G_1^{(1)}$ (resp. $G_2^{(1)}$). Note that $G_1^{(1)}$ (resp. $G_2^{(1)}$) may contain flow edges of half-flows rooted at v (resp. u). In the residual network, we orient a flow-edge in the opposite direction to the flow, irrespective of where the flow is rooted.

Since $G_1^{(1)}$ possibly contains parts of half-flow-paths rooted at v, some of the half-flow-paths for source-sink pairs routed in $G_1^{(1)}$ (after removing C) may also be mismatched after augmentation (see Fig 3), i.e. one of them is rooted at u and the other is rooted at v, even though both were routed from the single source u in $G_1^{(1)}$. The same also can happen for $G_2^{(2)}$.

Let M be the set of pairs of mismatched half-flow-paths that were routed after removing the edges of C. In any pair of mismatched half-flow-paths in M, at least one of them uses an edge of C. Hence, total number of mismatched half-flow-paths in M is at most p. We use the p u-v flow paths in $G^{(4)}$ to *correct* them, i.e., we obtain a complete flow path between s_i and t_i by using the two half-flow paths (ignoring direction) in $G^{(1)}$ and a path from u to v in $G^{(4)}$.

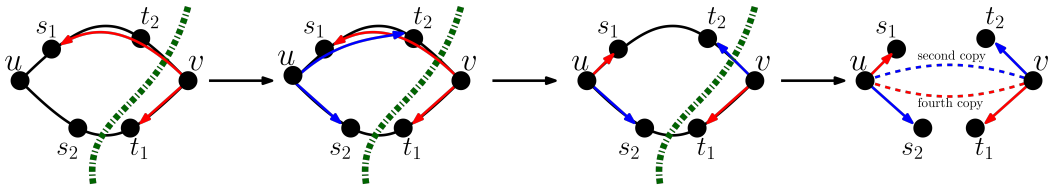


Figure 3 In the first picture (from left), we route a unit of flow from v to s_1 and v to t_1 and also pick a (u, v) cut (in green). This creates two connected components, one containing u and the other containing v. Observe that a part of the half-flow path from v to s_1 is also present in the component containing u. In the second picture, we augment a unit of flow from u to s_2 and u to t_2. This results in flow paths as shown in third figure, i.e. u to s_1, s_2 and v to t_1, t_2. Since any mismatched flow-path routed after picking the (green) cut has to cross an edge of the cut, they can be at most its capacity. As shown in last figure, we use one (u, v) flow path in the second copy to correct (s_1, t_1) flow and another (u, v) path in the fourth copy to correct (s_2, t_2) flow.

Similarly, we correct the p units of flow routed before deleting C by using at most p u-v flow paths in $G^{(2)}$. After these corrections we have as much resultant flow between the terminal pairs as the number of half-flow pairs routed. Note that we did not use $G^{(3)}$ yet, but we will need it for routing in the general case (see Section 7). Hence, we obtain a flow of congestion 3 in this case.

Let S_1 (resp. S_2) be the set of reachable vertices from u in $G_1^{(1)}$ (resp. from v in $G_2^{(1)}$) at the end of the algorithm (i.e. when we are not able to send any more flow in $G_1^{(1)}$ and $G_2^{(1)}$). We pick $C \cup \delta(S_1) \cup \delta(S_2)$ as our multicut. It is straightforward from construction that this is indeed a multicut. Hence the value of the multicut is $p + |\delta(S_1)| + |\delta(S_2)|$. Note that $|\delta(S_1)| + |\delta(S_2)|$ is at most the total number of half-flows routed as each edge in $\delta(S_1)$ (resp. $\delta(S_2)$) is saturated with flow going outside of S_1 (resp. S_2). Using the fact that p is at most the total number of half-flow pairs routed, we have that the value of the multicut is at most 3 times the total flow. It is easy to see that the run time of the above algorithm is similar to that of the Ford-Fullkerson algorithm, and hence we have the following theorem.

▶ **Theorem 4.** *Given an edge-capacitated parallel-path graph and source-sink pairs such that none of the source-sink pairs lie on one of the parallel paths, we can find an integral flow of value f with congestion 3, and a multicut of value at most $3f$ in time polynomial in size of the graph and the largest capacity. For unit capacity graphs, the algorithm runs in polynomial time.*

6 Augmenting External Flows into a Parallel-Path Graph

We showed in the previous section how to successfully augment multicommodity flows in a parallel-path graph H (with no terminal pairs on a path). Now, suppose H occurs as a building block of a series-parallel composition during the construction of a (larger) series-parallel graph. In our algorithm for series-parallel graphs, it is crucial that we are able to augment flows coming from vertices outside H into H through its merge vertices. Moreover, this has to be done in a way that the flows routed already inside do not get destroyed. We show in this section that a careful use of copies of the graph allows us to extend the augmenting paths framework of Ford and Fulkerson [7] to augment external flows into a parallel-path graph.

We first process all the source-sink pairs which are contained inside H using the algorithm described in Section 5. If a cut separating u and v in H is picked by the algorithm, then as shown in Section 5, we may safely continue to augment flow coming into $H^{(1)}$ by using

the augmenting paths algorithm. This is because the maximum number of mismatched half-flow-paths arising after a u-v cut is picked is at most the value of the minimum (u, v) cut and can be corrected by using one of the u-v flow paths in $H^{(4)}$. Also, mismatched half-flow-paths that were routed before the (u, v) cut was picked can be corrected using u-v flow paths in $H^{(2)}$.

We next show that we can safely continue to augment flows into $H^{(1)}$ from outside (i.e. for source-sink pairs not contained inside H) even in the other case i.e. when a (u, v)-cut has not been picked by the algorithm. As before, let p denote the value of minimum u-v cut in H. Suppose that a total flow less than p is routed by the algorithm. This implies that no (u, v)-cut is picked. Let the number of half-flow-paths incident at u, v be f_u, f_v respectively and the total flow be $f = f_u + f_v$. Let $H^{(1)}_w$ be the graph formed by adding a vertex w to $H^{(1)}$ and connecting it to u and v with edges of capacity f_{wu} and f_{wv} respectively. Suppose we are able to augment $f_w = f_{wu} + f_{wv}$ units of flow in $H^{(1)}_w$ from w in the residual graph (note that in the residual graph, all the flow paths in $H^{(1)}$ are rooted and directed away from u and v). Then we show how to use the additional $2f \leq 2p$ edge-disjoint paths from u to v, in $H^{(2)}$ and $H^{(3)}$ (f flow-paths in each) to reconstruct feasible flow paths, i.e. for each flow augmentation that happened from w to a vertex y, we produce a flow path from w to y, in addition to the flow paths that were already routed inside H.

▶ **Lemma 5.** *All flow paths (old and new) in $H^{(1)}_w$ can be reconstructed by using at most $2f$ u-v paths.*

Proof. To prove the lemma, we need the following crucial observation: if we augment a unit flow from the vertex w to x in $H^{(1)}_w$, then the amount of outgoing and incoming flow after augmentation remains unchanged for every vertex on the augmenting path except for w and x. The net flow (i.e. the amount of outgoing flow minus the incoming flow) of w increases by 1 while that of x decreases by 1. Let $(s_1, t_1), \ldots, (s_q, t_q)$ be the q source-sink pairs which were routed inside $H^{(1)}$ and $h_1, h_2, \ldots, h_q$ be the amount of flow routed for each one of them. Let $w_1, w_2, \ldots, w_l$ be the vertices to which we augmented flow from w in $H^{(1)}_w$ and $d_1, d_2, \ldots, d_l$ be the flow routed for each of them. Let $O = \{s_1, t_1, \ldots, s_q, t_q\}$ and $N = \{w_1, w_2, \ldots, w_l\}$. Before the augmentations from w, the net flow out of u and v in $H^{(1)}$ is f_u and f_v respectively and the net flow out of each vertex in O is $-h_i$. After the augmentations, the net flow out of u and v within $H^{(1)}$ (i.e. without taking into account flow on edges wu and wv) are $f_u + f_{wu}$ and $f_v + f_{wv}$ respectively, while that of vertices in O, N are $-h_i, -d_j$ respectively.

Since u and v have positive net flow in $H^{(1)}$, vertices in $O \cup N$ have negative net flow and rest of the vertices have zero net flow, we must have flow paths (with suitable flow value) from u, v to all the vertices in $O \cup N$. We first correct the flow paths corresponding to the source-sink pairs $(s_1, t_1), \ldots, (s_q, t_q)$ by using $\min(f_u, f_v) \leq f$ edge disjoint paths between u and v in $H^{(2)}$ and $H^{(3)}$. If exactly f_{wu} (resp. f_{wv}) edge disjoint paths starting at u (resp. v) terminate at vertices in N, then we already have a feasible flow. If $f_{wu}+g$ (resp. $f_{wv}-g$) units of flow incident at u (resp. v) terminate at vertices in N, then we use g flow paths from u to v to correct the flow paths originating at w. We now argue that $|g| \leq \max(f_u, f_v)$. This follows from the fact that $f_u - g$ (resp. $f_v + g$) paths incident at u (resp. v) must terminate in O, hence $|g| \leq f_u$ or $|g| \leq f_v$ which gives $|g| \leq \max(f_u, f_v)$. Hence total number of paths between u and v used to correct the flows is at most $\max(f_u, f_v) + \min(f_u, f_v) = f_u + f_v = 2f \leq 2p$. ◀

We will build on the intuition developed in this section to give a routing algorithm for the general case in the next section.

7 Routing Algorithm for Series-Parallel Graphs

Building upon the ideas developed in the previous sections, we now describe the full algorithm for routing flows in series-parallel graphs. We will also pick some cut-edges during the routing here, but they will not form the whole multicut; the algorithm for picking the complete multicut will be presented later in Section 9. Our routing algorithm is recursive using the recursive construction of series-parallel graphs through series and parallel compositions.

Let G be the input series-parallel graph and let u and v be its merge vertices. We construct four copies of G denoted by $G^{(1)}, G^{(2)}, G^{(3)}$ and $G^{(4)}$, each with the same capacities as G. The algorithm outputs the following: a set of (directed) flow paths F in $G^{(1)}$, a set of cut-edges C (not necessarily a multicut), and two numbers $(l^{(2)}, l^{(4)})$. During the algorithm we will *reserve* some flow-paths between the merge vertices u and v in $G^{(2)}, G^{(3)}$, and $G^{(4)}$ for *flow correction*. The *reserved flow* will be used in the flow-correction phase in Section 8 to correct the *mismatched flows* in $G^{(1)}$. The number $l^{(2)}$ gives the number of flow paths available in each of $G^{(2)}$ and $G^{(3)}$ between u and v for flow-correction in the future, after we have *reserved* the flow-paths for correcting the flows routed so far. The number $l^{(4)}$ gives the same for $G^{(4)}$. In a sense, $l^{(2)}, l^{(3)}, l^{(4)}$ are the *residual flow-correcting capacities* that G passes on up to its parent in the recursion call.

We also maintain a global tuple $D = \{d_1, d_2, \ldots, d_k\}$ where d_i denote the amount of flow routed for terminal pairs (s_i, t_i). We will maintain throughout the algorithm that $d_i = f^-(s_i) = f^-(t_i)$, where $f^-(x)$ denote the incoming flow to x in F. Whenever we augment a new unit of flow for an s_i-t_i pair, we assume that d_i increases by one, even if not mentioned explicitly.

We first describe the **base case**, i.e. if G is an edge (u, v) with capacity $c(u, v)$. If (u, v) do not form a source-sink pair, then the algorithm returns an empty flow, $l^{(2)} = l^{(4)} = c(e)$ and $C = \emptyset$. If (u, v) is a source-sink pair i.e. if $(u, v) = (s_i, t_i)$, we send $c(e)$ units of (directed) flow from u to v in $G^{(1)}$, reserve $c(e)$ amount of flow-paths from u to v in each of $G^{(2)}, G^{(3)}, G^{(4)}$ and return $l^{(2)} = l^{(4)} = 0$ and $C = \{(u, v)\}$.

Now, we go to the recursion step. Let G be composed of G_1 and G_2 in series or parallel. Let u_1, v_1 be the merge vertices of G_1 and u_2, v_2 be the merge vertices of G_2. We first run the routing algorithm on G_1 and G_2 separately. For $i = 1, 2$, let $(F_i, l_i^{(2)}, l_i^{(4)}, C_i)$ be the output of the algorithm. Depending on whether G_1 and G_2 are joined in series or parallel, the algorithm now branches out into two cases.

7.1 Parallel Case

Recall that in the parallel case, G is obtained by connecting G_1 and G_2 in parallel i.e. by setting $u = u_1 = u_2$, and $v = v_1 = v_2$. Before routing flow, we remove all the edges in C_1 and C_2 from $G^{(1)}$. Our algorithm here is similar to the parallel-path case in Section 5. We say that a terminal pair is *newly connected* if one of the terminals is in G_1 and the other is in G_2. If no source-sink pairs get newly connected due to the parallel combination, we simply return $F_1 \cup F_2$, $l^{(i)} = l_1^{(i)} + l_2^{(i)}$ for $i = 2, 4$ and $C = C_1 \cup C_2$.

Otherwise, some source-sink pairs get newly connected. All paths between the newly connected source-sink pairs have to contain either u or v. Let s be the vertex obtained by identifying u and v as a single vertex. We initialize the flow F to be $F_1 \cup F_2$. Let R_s be the set of reachable vertices from s in the residual graph of $G^{(1)}$ with respect to the flow F. We say that a newly connected source-sink pair (s_j, t_j) is **reachable** from s if both $s_j \in R_s \ \ \& \ \ t_j \in R_s$.

If there is such a reachable newly connected source-sink pair then we augment in F, one unit of flow each to s_j and t_j from the vertex s and set $d_j = d_j + 1$. Since $s_j \in G_1$ and $t_j \in G_2$, the two augmenting paths from the vertex s to s_j and t_j are vertex disjoint except

at s. Hence we can augment along both the paths simultaneously. However, note that this does not mean we can directly construct a flow path between s_j and t_j by combining both of these half-flows (ignoring directions), as the half-flow path to s_j may begin at u while the half-flow path to t_j may begin at v. Later in the correction step in Section 8, we will use a (u, v) path in either $G^{(2)}, G^{(3)}$, or $G^{(4)}$ to obtain a feasible flow.

As in the parallel-path case, we repeat the above routing procedure until one of the following happens: either there are no more reachable source-sink pairs from s, or we have routed $l_1^{(2)}+l_2^{(2)}$ many units. Let f denote the number of half-flow pairs routed after connecting G_1, G_2 in parallel.

1. In case 1, i.e. if the routing terminates with $f < l_1^{(2)} + l_2^{(2)}$, then we reserve f out of the available $l_1^{(2)} + l_2^{(2)}$ u-v paths for flow correction in each of $G^{(2)}$ and $G^{(3)}$. We return the flow F, cut edges $C = C_1 \cup C_2$, and numbers $l^{(2)} = l_1^{(2)} + l_2^{(2)} - f$, and $l^{(4)} = l_1^{(4)} + l_2^{(4)}$.
2. In case 2, i.e. if $f = l_1^{(2)} + l_2^{(2)}$, then we pick a min-cut separating u and v in $G^{(1)}$, say C_s. We set $C = C_1 \cup C_2 \cup C_s$. Let $G_u^{(1)}$ and $G_v^{(1)}$ be the two graphs formed after removing the edges of C_s from $G^{(1)}$. Even after removing the cut edges in C_s, there might be source-sink pairs that are reachable from u in $G_u^{(1)}$ or v in $G_v^{(1)}$. We augment F by routing from u in $G_u^{(1)}$ (resp. from v in $G_v^{(1)}$) to the reachable source-sink pairs and update D accordingly. We do this until no source-sink pairs are reachable from u in G_u and v in G_v. We reserve $l_1^{(2)} + l_2^{(2)}$ u-v paths in each of $G^{(2)}$ and $G^{(3)}$ and $l_1^{(4)} + l_2^{(4)}$ u-v paths in $G^{(4)}$ for flow corrections and return $l^{(2)} = l^{(4)} = 0$ along with the flow F and cut $C = C_1 \cup C_2 \cup C_s$.

7.2 Series Case

Recall that in the series case, G is obtained by connecting G_1 and G_2 in series, i.e. by identifying $w = v_1 = u_2$. Before routing flow, we remove all the edges in C_1 and C_2 form the first copy of G. To make the presentation simpler, w.l.o.g we assume that $l_1^{(2)} \leq l_2^{(2)}$.

If no new source-sink pairs get connected due to the series combination, we simply return $F_1 \cup F_2$, $l^{(i)} = \min\{l_1^{(i)}, l_2^{(i)}\}$ for $i = 2, 4$ and $C = C_1 \cup C_2$.

Otherwise, some new source-sink pairs get connected. All paths between the newly connected source-sink pairs have to contain w. Nevertheless we route from any of u_1, w and v_2 as below. We identify u_1, w and v_2 into a super-source vertex, say s, and find a source-sink pair (s_j, t_j) such that both s_j and t_j are reachable from s in the residual graph of $G^{(1)}$ with respect to flow F, which is initialized to $F_1 \cup F_2$. We call such source-sink pairs **reachable** from s. Note that if both are reachable then both can be routed simultaneously, as one of them lies in G_1 and the other in G_2. We augment in F one unit of flow from s to s_j and from s to t_j and update D accordingly. However, note that this might not directly give us a flow path from s_j to t_j, as the half-flow path to s_j may begin at u_1 while the half-flow path to t_j may begin at v_2. Later in the correction step, we will use a (u_1, v_2) path in $G^{(2)}, G^{(3)}$, or $G^{(4)}$ to obtain a feasible flow.

We keep augmenting as above until one of the following happens: either no more source-sink pairs are reachable from s or we have routed $\min\{l_1^{(2)}, l_2^{(2)}\}$ units of flow. Let f denote the total source-sink flow routed after connecting G_1, G_2 in series.

1. In case 1, i.e. if the routing terminates with $f < \min\{l_1^{(2)}, l_2^{(2)}\}$, then we reserve f flow paths between u_1-v_2 in $G^{(2)}$ and $G^{(3)}$ (note that these reserved flow-paths goes through w and in the flow-correction phase, we may use such a flow-path to correct a flow between u_1 and w, or w and v_2, or u_1 and v_2). We return the flow F, the cut $C = C_1 \cup C_2$, and $l^{(2)} = \min\{l_1^{(2)}, l_2^{(2)}\} - f$, and $l^{(4)} = \min\{l_1^{(4)}, l_2^{(4)}\}$.

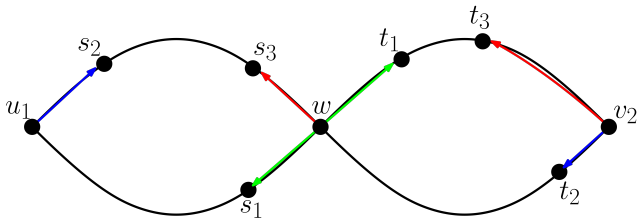
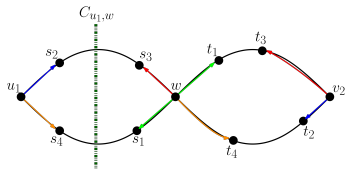
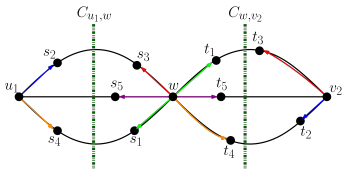


Figure 4 On the left part (u_1, w), we have $l_1^{(2)} = 4$ and on the right part, (w, v_2) we have $l_2^{(2)} = 5$.

2. In case 2, i.e. if $f = \min\{l_1^{(2)}, l_2^{(2)}\} = l_1^{(2)}$(w.l.o.g), then we pick a min-cut separating u_1 and w in $G^{(1)}$, say $C_{u_1,w}$. We set $C = C_1 \cup C_2 \cup C_{u_1,w}$ and $G^{(1)}_{u_1}$ and $G^{(1)}_{w,v_2}$ be the two graphs formed after removing the edges of C from $G^{(1)}$. Let s' be the vertex formed by identifying w, v_2 as a single vertex. Even after removing the cut edges in $C_{u_1,w}$, there might be source-sink pairs that are reachable from s' in $G^{(1)}_{w,v_2}$. We augment flow (in F) from s' in $G^{(1)}_{w,v_2}$ to the reachable source-sink pairs from s' until one of the following happens: either no more source-sink pairs are reachable from s' or we have augmented $l_2^{(2)} - l_1^{(2)}$ units of such flow.
   a. In case a), i.e. if no more terminal pairs are reachable from s' and $f < l_2^{(2)}$ (here f is the total amount of flow augmented after connecting G_1 and G_2 in series), then we reserve $f - l_1^{(2)}$ units of flow paths between w and v_2 in $G^{(2)}$ and $G^{(3)}$, reserve $l_1^{(4)}$ flow-paths between u_1 and w in $G^{(4)}$, and return the flow F, $l^{(2)} = l^{(4)} = 0$ and $C = C_1 \cup C_2 \cup C_{u_1,w}$.
   b. In case b), i.e. if $f = l_2^{(2)}$ (i.e. $l_1^{(2)}$ units of flow was routed before deleting $C_{u_1,w}$ and $l_2^{(2)} - l_1^{(2)}$ units of flow afterwards), then we pick a min-cut separating w and v_2 in $G^{(1)}_{w,v_2}$, say C_{w,v_2}. We set $C = C_1 \cup C_2 \cup C_{u_1,w} \cup C_{w,v_2}$ and let $G^{(1)}_w$ and $G^{(1)}_{w,v_2}$ be the two graphs formed after removing the edges of C from $G^{(1)}_{w,v_2}$. Even after removing the cut edges in C_{w,v_2}, there might be source-sink pairs that are reachable from w in $G^{(1)}_w$. We augment flow (in F) from w in $G^{(1)}_w$ to the reachable source-sink pairs from w. We do this until no source-sink pairs are reachable from w in $G^{(1)}_w$. We reserve $l_1^{(2)} + l_2^{(2)}$ amount of $w - v_2$ flow-paths in $G^{(2)}$ and $G^{(3)}$. We also reserve $l_1^{(4)}$ amount of $u_1 - w$ flow and $l_2^{(4)}$ amount of $w - v_2$ flow in $G^{(2)}$. We return $F, l^{(2)} = l^{(4)} = 0$, and $C = C_1 \cup C_2 \cup C_{u_1,w} \cup C_{w,v_2}$.



(a) Case 2a.



(b) Case 2b.

8 Constructing Feasible Flows

Let $D = \{d_1, d_2, \ldots, d_k\}$ be the vector of all the source-sink flow values at the end of the algorithm. We will show that a feasible flow between the terminal pairs of value $\sum_{i=1}^{k} d_i$ can be constructed using the second, third and fourth copy of G. First we note some of the properties of the routing algorithm, which will be helpful in proving further results.

Let G be a series-parallel graph with merge vertices (u, v). Let $(F, l^{(2)}, l^{(4)}, C)$ be the output of the algorithm on G and let f be the total flow (i.e. the number of half-flow pairs routed) in F. When G is not an edge we will assume that G is formed by the series or parallel composition of G_1 and G_2 with merge vertices (u_1, v_1) and (u_2, v_2) respectively. Let $(F_1, l_i^{(2)}, l_i^{(4)}, C_i)$ denote the output of the algorithm for G_i for $i = 1, 2$.

▶ **Lemma 6.** *After G has been processed, there exists a (u, v)-cut of value at most $l^{(4)}$ in $G \setminus C$.*

▶ **Lemma 7.** $c(C) \leq 2f$.

Recall that while routing in G, we reserved some flow paths between u and v for flow corrections. The next claim shows that the total value of reserved flow paths (across all iterations) is at most four times the total value of flow routed in G.

▷ Claim 8. The value of reserved flow paths in each of $G^{(2)}$ and $G^{(3)}$ is at most f and that in $G^{(4)}$ is at most $2f$.

8.1 The Augmentation Property and Flow Correction

We now show that a feasible flow of value equal to the total augmented flow can be obtained by using the reserved flow paths, at each stage of the algorithm. To prove this result, we inductively maintain an invariant called as the **augmentation property**, specified below. Let G^* be the final graph and G be the graph obtained at an intermediate stage. Let (u, v) be the merge vertices of G. For giving the augmentation property, we distinguish between two cases, depending on whether a cut separating (u, v) has been picked by the algorithm so far. In both cases the augmentation property states that we can reconstruct all the source-sink flow paths that were augmented inside G (i.e. all the flow paths augmented inside G before its processing is finished), using only the flow paths reserved in the copies of G. In addition, to this, the property also states the following depending on the case.

- Case 1. No (u, v) cut has been picked by the algorithm so far: suppose a flow of $f_1, f_2, \ldots, f_k$ was augmented to (terminal) vertices $t_1, t_2, \ldots, t_k$ after the processing of G was finished (these are external flows that come from outside of G). Furthermore, suppose that f_u and f_v units of flow was augmented from u and v respectively into G (by external flows) after the processing of G is finished, i.e. $f_u + f_v = \sum_{i=1}^{k} f_i$. Then the augmentation property states that we can additionally reconstruct these flow paths using only the reserved paths in copies of G such that: (i) exactly f_u (resp. f_v) units of flow path emerge from u (resp. v) (ii) there is exactly f_i units of incoming flow incident at each t_i. In other words, we reconstruct all flow paths corresponding to augmenting paths, except that they might originate from either u or v (there might have been a path originally augmented from u to t_i, but in the reconstructed paths the path to t_i might be from v).
- Case 2. A (u, v) cut has been picked by the algorithm: let G_u, G_v be the two connected components of G (after deleting the cut edges) containing the vertices u, v respectively. Suppose a flow of value $f_1, f_2, \ldots, f_k$ was augmented into G_u (via u) to (terminal) vertices $t_1, t_2, \ldots, t_k$ after the processing of G was finished. Then the augmentation property states that we can reconstruct feasible flow paths (in addition to the source-sink flow paths that were augmented inside G before its processing was finished) using only the reserved flows for G, such that there is a flow of value f_i from u into t_i for each $i = 1, 2, \ldots, k$. The same holds true for G_v as well.

We show the following lemma by using induction on the structure of series-parallel graphs. This also implies that there exists a feasible flow of value $\sum_{i=1}^{k} d_i$.

▶ **Lemma 9.** *For any graph G obtained during an intermediate stage of the routing algorithm, the augmentation property holds.*

9 Picking a Multicut

Let C be the cut edges picked after the completion of routing phase for G. In this section, we assume that the edges of C have been removed from G. In addition to the cut edges C, we pick another set of edges Y such that $C \cup Y$ is a feasible multicut for the given instance. We say that the edges in C were picked during the phase 1 of the algorithm. We now describe the phase 2 of the algorithm, where we pick the edges in Y. We start with all the vertices of G as unmarked and initialize the set Y as empty. We process the nodes of a tree-decomposition T of G (with treewidth 2) in a top-down manner, i.e. we process a node only after all its ancestors are processed. Let X be the current node we are processing. Recall that each node X corresponds to a series or a parallel combination of two subgraphs of G and it consists of union of the merge vertices of these two subgraphs. Let C_X be the set of reachable vertices in the residual graph of G_X, from X, just after the processing of X in phase 1 has been completed. Recall that the residual graph arises w.r.t to the current (directed) flow in the first copy of the graph. If all the vertices in X are already marked then do nothing. Let X' be the set of unmarked vertices in X. For any vertex x, let $\texttt{Comp}_G(x)$ denote the connected component containing x in the current graph (i.e. $G \setminus (Y \cup C)$). For each $x \in X'$, mark all the vertices in $C_X \cap \texttt{Comp}_G(x)$, add the edges in $Y_X := \delta(C_X) \cap E(\texttt{Comp}_G(x))$ to Y, and delete those edges from G. Repeat this process until all the vertices of G have been marked. Then the union of Y and C is our required multicut.

▶ **Lemma 10.** *Let X' be a node of T and X be a node of $T_{X'}$. Then, $C_{X'} \cap V(G_X) \subseteq C_X$.*

Proof. Since any path in the residual graph from outside G_X has to enter through X and all edges in $\delta(C_X)$ are directed inwards to C_X in the residual graph, the vertices in $V(G_X) \setminus C_X$ can never become reachable from any vertex outside G_X in the residual graph. The lemma follows from this easily. ◀

▶ **Lemma 11.** *$C \cup Y$ is a multicut of G for the given terminal-pairs.*

Proof. Suppose $Y \cup C$ does not cut some terminal pair s, t. This means $G - Y - C$ contains a path P between s and t. Let X be the bottom-most node in T such that G_X contains both s and t. Clearly P contains at least one vertex from X. Let this vertex be x. We branch into 2 cases depending on when x was marked in phase 2.

In Case 1, we suppose x was marked during the processing of X. Without loss of generality we can assume that the sub-path of P between x and s contains an edge of $\delta_{G-C}(C_X)$, say e (follows from phase 1 algorithm). Since e is in the same connected component as x in $G - C - Y$, and $e \in C_X$, we have that e would have been picked into Y during the processing of X, a contradiction.

In Case 2, we suppose x was marked before the processing of X. Let X' be the node during whose processing, x was marked. Clearly X is in $T_{X'}$. Thus, by Lemma 10, we have that $C_{X'} \cap V(G_X) \subseteq C_X$. Hence, without loss of generality we can assume that the sub-path of P between x and s contains an edge of $\delta_{G-C}(C_{X'})$, say e. Since e is in the same connected component as x in $G - C - Y$, and $e \in C_{X'}$, we have that e would have been picked into Y during the processing of X'. ◀

For a node X of T, let $f(X)$ denote the number of half-flow paths introduced during the processing of X in phase 1. Since every flow path consists of two half flow paths, we have that total flow routed in phase 1, $f = \sum_{X \in T} f(X)/2$. For a node X of T, let $r(X)$ denote the number of flow paths reserved between the vertices of X when T_X was being processed in phase 1. From Claim 8, it follows that $\sum_{X \in T} r(X) \leq \frac{4}{2} \cdot \sum_{X \in T} f(X) \leq 2 \cdot \sum_{X \in T} f(X)$.

Let $M(X)$ denote the set of previously unmarked vertices that becomes marked during the processing of X in Phase 2. Let $I(X)$ denote the set of nodes of T that have non-empty intersection with $M(X)$.

▶ **Lemma 12.** *For a node X of T, the total capacity of edges picked into the cut Y during the processing of X in Phase 2 is at most $\sum_{X' \in I(X)} f(X') + \sum_{X' \in I(X)} r(X')$.*

▶ **Lemma 13.** *For a node X' of T, the number of nodes X of T such that $X' \in I(X)$ is at most 3.*

Proof. During the processing in Phase 2 of each X such that $X' \in I(X)$, at least one unmarked vertex in X' becomes marked. The lemma follows as there are at most 3 vertices in X. ◀

▶ **Lemma 14.** *$|Y \cup C|$ is at most 20 times the amount of flow routed between the terminal pairs by our algorithm.*

Proof. Recall that $\sum_{X \in T} r(X) \leq 2 \cdot \sum_{X \in T} f(X)$. From Lemma 13 and Lemma 12, it follows that $|Y|$ is at most $3 \cdot (\sum_{X \in T} f(X) + 2 \cdot \sum_{X \in T} r(X)) \leq 9 \cdot \sum_{X \in T} f(X)$. Hence, the total capacity of edges in Y is at most 18 times the total flow routed in phase 1. From Lemma 7, we have that $|C|$ is at most twice the total flow routed by the phase 1 algorithm. Therefore, total capacity of edges in $Y \cup C$ is at most 20 times the total flow routed by the phase 1 algorithm. ◀

This concludes our main result Theorem 2 and also implies the following corollary.

▶ **Corollary 15.** *Let G be an undirected, (integer) edge capacitated treewidth-2 graph and $\{(s_i, t_i)\}_{i=1}^{k}$ be the source-sink pairs. Our algorithm gives an 80-approximation for computing a multicut w.r.t. the source-sink pairs.*

References

1. Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 79–88, 2014.
2. Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of computing*, 2(1):137–146, 2006.
3. Chandra Chekuri, Guyslain Naves, and F Bruce Shepherd. Maximum edge-disjoint paths in k-sums of graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 328–339. Springer, 2013.
4. Julia Chuzhoy and Shi Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 233–242. IEEE, 2012.
5. Denis Cornaz. Max-multiflow/min-multicut for G+H series-parallel. *Discret. Math.*, 311(17):1957–1967, 2011. doi:10.1016/j.disc.2011.05.025.
6. Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On routing disjoint paths in bounded treewidth graphs. *arXiv preprint*, 2015. arXiv:1512.01829.

7 Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. In *Classic papers in combinatorics*, pages 243–248. Springer, 2009.

8 Naveen Garg and Nikhil Kumar. Dual half-integrality for uncrossable cut cover and its application to maximum half-integral flow. In *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173, page 55. Schloss Dagstuhl – Leibniz-Zentrum for Informatik, 2020.

9 Naveen Garg, Nikhil Kumar, and András Sebö. Integer plane multiflow maximisation: Flow-cut gap and one-quarter-approximation. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 144–157. Springer, 2020.

10 Naveen Garg, Vijay V Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi) cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

11 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

12 T Chiang Hu. Multi-commodity network flows. *Operations research*, 11(3):344–360, 1963.

13 Philip Klein, Serge A Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 682–690. ACM, 1993.

14 Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is np-complete for series–parallel graphs. *Discrete Applied Mathematics*, 115(1-3):177–186, 2001.

15 Loïc Seguin-Charbonneau and F Bruce Shepherd. Maximum edge-disjoint paths in planar graphs with congestion 2. In *IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 200–209. IEEE, 2011.

16 Eva Tardos and Vijay V Vazirani. Improved bounds for the max-flow min-multicut ratio for planar and kr, r-free graphs. *Information Processing Letters*, 47(2):77–80, 1993.

17 Hui Zang, Jason P Jue, and Biswanath Mukherjee. A review of routing and wavelength assignment approaches for wavelength-routed optical wdm networks. *Optical networks magazine*, 1(1):47–60, 2000.

A Figures

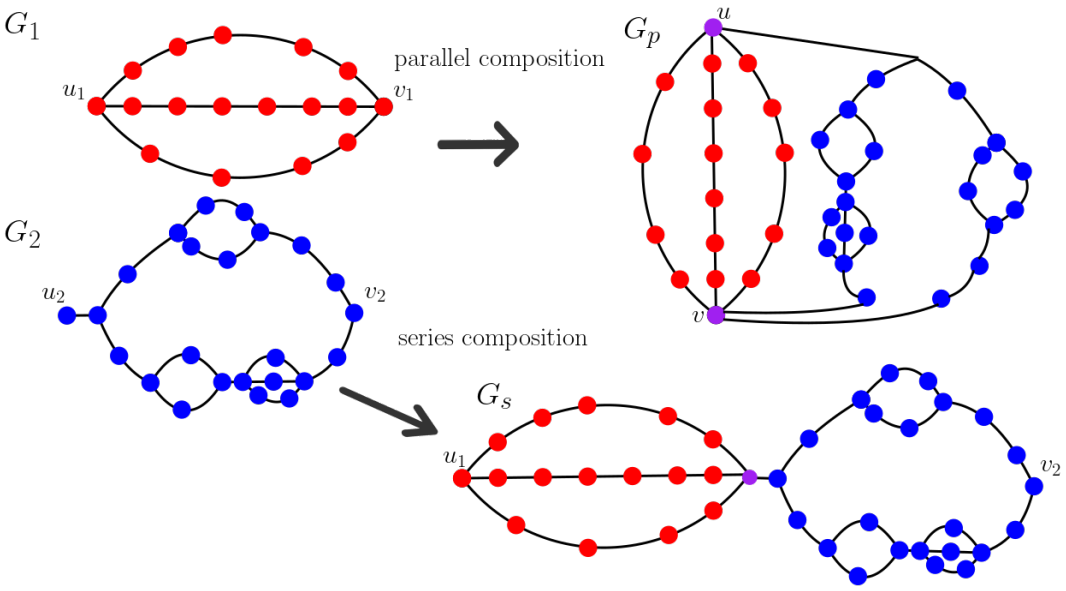


Figure 6 Series and Parallel Compositions.

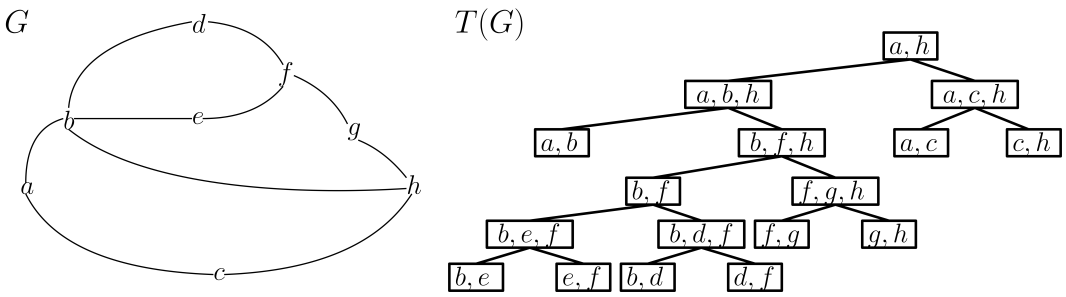


Figure 7 Series-Parallel Tree-Decomposition.