Maximum Edge-Disjoint Paths in Planar Graphs with Congestion 2

Loïc Séguin-Charbonneau
Department of Science
Royal Military College Saint-Jean
St. Jean sur Richelieu, QC, Canada
loicseguin@gmail.com

F. Bruce Shepherd
Mathematics and Statistics
McGill University, Montreal QC
and Theory Group
Microsoft Research, Redmond, WA
bruce.shepherd@mcgill.ca

Abstract— We study the maximum edge-disjoint path problem (MEDP) in planar graphs. We are given a set of terminal pairs and wish to find a maximum *routable* subset of demands. That is, a subset of demands that can be connected by edge-disjoint paths. It is well-known that there is an integrality gap of order square root of the number of nodes for this problem even on a grid-like graph, and hence in planar graphs (Garg et al.). In contrast, Chekuri et al. show that for planar graphs, if LP is the optimal solution to the natural linear programming relaxation for MEDP, then there is a subset of size OPT over the logarithm of the number of nodes which is routable with congestion 2. Subsequently they showed that it is possible to get within a constant factor of the optimal solution with congestion 4 instead of 2. We strengthen this latter result to show that a constant approximation is possible also with congestion 2 (and this is tight via the integrality gap grid example). We use a basic framework from work by Chekuri et al. At the heart of their approach is a 2-phase algorithm that selects an Okamura-Seymour instance. Each of their phases incurs a factor 2 congestion. It is possible to reduce one of the phases to have congestion 1. In order to achieve an overall congestion 2, however, the two phases must share capacity more carefully. For the Phase 1 problem, we extract a problem called *rooted clustering* that appears to be an interesting problem class in itself.

Keywords-Network flows, edge-disjoint paths, confluent flows, clustering

1. Introduction

We consider the maximum (undirected) edge-disjoint path problem (MEDP) in planar graphs. MEDP is formulated as follows. We are given a planar undirected graph $G = (V, E)$ and a set of node pairs (*demands*) $\mathcal{D} = \{s_1t_1, s_2t_2, \ldots, s_kt_k\}$. Define $X = \{s_1, s_2, \ldots, s_k, t_1, t_2, \ldots, t_k\}$. The nodes in X are called *terminals* and the two terminals in a pair are called *siblings*. We can typically assume that the terminals are distinct and thus the demands form a matching. For $v \in X$ we denote its sibling by $\sigma(v)$. We call a subset $S \subseteq \mathcal{D}$ *routable* if there is a collection of edge-disjoint paths joining the pairs in S. The objective of MEDP is to find a maximum routable subset of $\mathcal{D}$. More generally, each edge e may have an integer capacity $u(e)$, and we seek a collection of paths such that each edge is contained in at most $u(e)$ paths. For such a solution, if it includes a path joining the demand pair s_it_i, then the demand is said to be *routed* in the solution.

Consider the natural linear programming formulation (LP) for MEDP:

$$\begin{aligned} \max \sum_{i=1}^{k} x_i \quad \text{s.t.} & & & \qquad (1)\\ x_i - \sum_{P \in \mathcal{P}_i} f(P) &= 0 & 1 \le i \le k \\ \sum_{P : e \in P} f(P) &\le u(e) & \forall e \in E \\ x_i, f(P) &\in [0,1] & 1 \le i \le k, P \in \mathcal{P} \end{aligned}$$

where $\mathcal{P}_i$ is the set of paths between s_i and t_i and $\mathcal{P} = \cup_{i=1}^{k} \mathcal{P}_i$ is the set of all paths in G. Let OPT denote its optimal value. It is well-known that (1) may have a $\Omega(\sqrt{n})$ integrality gap [9] in the worst case, even in undirected planar graphs as shown by Fig. 1.

Note that for the grid example, one can actually route all demands if each edge may be used twice. Given the large integrality gap, it becomes natural to inquire about the affect of *low congestion routings* on approximability. An α*-congested* solution corresponds to a subset of demands routed via a path collection $\mathcal{P}$ where each edge e is contained in at most α (or $\alpha u(e)$) paths in the collection. It is well-known that $\Omega(\text{OPT})$

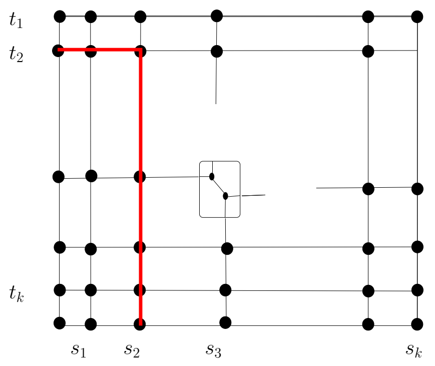

Figure 1. An $\Omega(\sqrt{n})$ integrality gap. Each node is split into two, as depicted.

may be achieved in a general graph with $O(\log n)$-congestion via randomized rounding [16]. In a recent advance, [1] found a polylogarithmic approximation for MEDP using $O(poly \log \log n)$ congestion. The focus of the present paper is the approximability of MEDP with $O(1)$ congestion in planar graphs. Kleinberg and Tardos, in their study of grid-like planar graphs [13], implicitly suggested that a polylogarithmic approximation may be possible with congestion 2 in planar graphs; they directly ask whether an $O(\log n)$-approximation is possible in Eulerian planar graphs with congestion 1. In [3], an $O(\log n)$ approximation is given for planar graphs if edge *congestion* 2 is allowed. This framework was later employed in [14] to get an $O(\log^2 n)$ approximation with congestion 1 for Eulerian planar graphs; this was strengthened to $O(\log n)$ in [12]. A different approach was ultimately devised [5] to prove that there is a constant factor approximation in planar graphs if congestion 4 is allowed. It was left open whether a constant factor approximation was possible with at most congestion 2 however. Due to the grid example, this would be a tight result. As our main result we resolve this question. Our approximation is with respect to the above multiflow LP relaxation:

Theorem 1.1. *If G is planar, and all edge capacities are at least 2, then the LP (1) for* MEDP *has a constant integrality gap. Moreover, there is a polytime algorithm which converts a fractional solution for* MEDP *with total flow F, into an integral congestion 2 solution with total flow $\Omega(F)$.*

We now give a very high level view of the algorithm from [5] which we refer to as PLANE-EDP4. More details are given in Section 3. After some standard reductions (e.g., to bounded degree graphs, and matching demands), PLANE-EDP4 repeatedly finds "sparse" cuts. Each cut breaks the graph into two planar parts G_1, G_2 (where G_1 is chosen to be minimal in a certain sense). The flow lost between the two parts (i.e., flow crossing the cut) is then "charged" to the flow within G_1. They then apply a one-time procedure to extract a large integrally-routed set of demands within G_1. After this, they recurse on G_2. Thus the meat of the algorithm is within the one-time routing procedure. This part consists primarily of a two-phase algorithm which converts a fractional routing into an integral one with congestion 4. This subroutine has a *clustering phase* and a *routing phase*. A factor of 2 congestion arises in each phase. Using an improved clustering scheme based on confluent flows, we are able to improve Phase 1 to only incur congestion 1 – Section 3. In order to achieve an overall congestion 2, however, Phase 1 must carefully lend some capacity to the second phase – Section 4.1. One may extract a stand-alone problem from the Phase 1 analysis. We call this "rooted clustering", and it is somehow related to both unsplittable and confluent flows – see Section 2. We briefly discuss this subproblem now.

A key step in many approximation algorithms involves grouping of some weighted terminals s_i, d_i in a graph into so-called *clusters*. For instance, in [11], [2], [4] a collection of edge-disjoint connected subgraphs are sought such that each subgraph contains $\Theta(1)$ of demand. Such clusters are usually easy to construct greedily from a spanning tree. In [5], however, there is an additional requirement that we are also given a face boundary C in the planar graph, and each cluster should contain a path to this face. Naturally, C could be identified with a single "sink" node t, and so with this additional requirement, we refer to this as a *rooted clustering* problem. A rooted clustering with edge congestion 2 is obtained in [5]. Unfortunately, we show that edge-disjoint clusterings of this type are not possible for directed graphs. Instead we show that if we are only concerned with "capturing" a constant fraction of demands between pairs, say s_i, t_i, then such an edge-disjoint rooted clustering can be found. These results are described in Section 2 and are obtained by modifying a confluent flow algorithm in [6].

2. Rooted Clustering

In this section we consider a problem where one must *groom* demands into "bundles" which then share a single path to some fixed destination node. The problem has common features with the single-sink unsplittable multiflow problem [7], but solutions to our grooming problem must satisfy additional requirements which makes them (seemingly) less straightforward. We derive one result (Theorem 2.3) used later in the disjoint path algorithm of Section 3.

In the *rooted clustering problem* we are given a graph $G = (V, E)$ (not necessarily simple, and either directed or undirected) with a specified *root* node $t \in V$. We are also given *terminals* $s_1, \ldots, s_k$ each with a *demand* $d_i \in [0, 1]$. A *(unsplittable) rooted clustering* is a collection of connected subgraphs (called *clusters*) $H_1, \ldots, H_c$ each containing t, and an assignment $f : \{s_1, \ldots, s_k\} \to \{H_1, \ldots, H_c\}$. In other words, if $f(s_i) = H_j$, then terminal s_i is said to be *assigned* to cluster H_j; we also say that s_i is *covered* by H_j. In addition we have:

$$\sum_{s_i \text{ assigned to } H_j} d_i = O(1)$$

(note that in some cases, we need the sum to be $\Theta(1)$; in either case, we are not concerned with the hidden constants). The *congestion* of a clustering is the maximum number of times that any edge appears in the list of clusters. If the congestion is 1, then we refer to it as an *edge-disjoint* rooted clustering. Note that a node (including any terminal) may appear in several clusters. However, we require that demand from each terminal is assigned to a single cluster. This condition can be relaxed to obtain a *splittable* version of the problem, where a demand of d_i may be split across multiple clusters. We remark that the existence of an (unsplittable) edge-disjoint rooted clustering implies the existence of a single-sink unsplittable flow for the demands d_i, with "congestion" at most the maximum total demand of a cluster (cf. [7] for definitions and related work). In fact, rooted clustering seems to live "between" unsplittable flows and confluent flows as we see later.

We call an instance *edge-normalized* if there is a network flow that routes demand d_i from each s_i to t, and sends flow of at most 1 on each edge of G. Since we study single-sink instances (rooted clustering) this simply means that the natural cut condition holds. Node-normalization is defined similarly. Clearly any node-normalized instance is (edge-) normalized. As with unsplittable and confluent flows, one often assumes that an instance is normalized, since it is an obvious necessary condition for a rooted clustering to exist.

In [5], the following is (essentially) shown.

Theorem 2.1 ([5]). *If G is bounded degree, and G, t, s_i, d_i is an edge-normalized instance, then there is a congestion 2 rooted clustering of the demands.*

A natural goal is to try to strengthen this to find edge-disjoint rooted clusters. Unfortunately, this is not possible, at least for directed graphs. This is shown by the following example where the bottom layer nodes are all adjacent to the root.[1] In the directed setting, consider a cluster containing the top node. The circled nodes at layer i all have demands $1/i$. Since the cluster must contain a directed path P to the root, and all arcs are directed downwards, every terminal on this path must actually be contained in the same cluster if we require edge-disjointness. Hence this cluster includes at least one terminal of demand $\frac{1}{i}$ at each level i, and hence it has a total demand of $\sum_{i=1}^{k} \frac{1}{i} = \Omega(\log n)$, which is too large.

We thus focus on so-called incomplete clusterings, which will be sufficient for our application to disjoint paths.

2.1. Incomplete and Partial Clustering

To avoid the difficulty of the previous example, we relax the condition that we have a *complete* clustering, where each terminal is assigned to some cluster. An *incomplete* clustering is thus one where some demands

[1]This example is essentially given in [6] as an example of a $\Omega(\log n)$ gap for the congestion minimization LP for confluent flows.

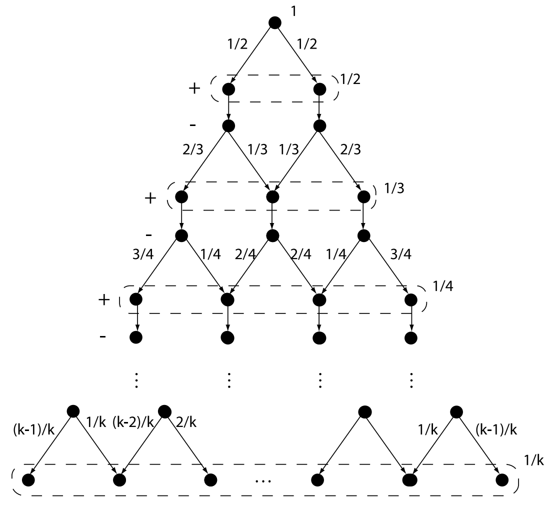

Figure 2. There is no arc-disjoint rooted total clustering (splittable or unsplittable).

may not be assigned to a cluster; demands that are assigned are said to be *covered* by the clustering. We show that one can always cluster a large fraction of demands with edge-disjoint clusters.

Theorem 2.2 (Unsplittable Incomplete Clustering). *For any $\kappa \geq 1$ and any edge-normalized instance, there is an edge-disjoint rooted clustering that covers at least $\frac{\kappa-1}{2\kappa(\kappa+2)}\Delta$ of the total demand $\Delta = \sum_i d_i$.*

To maximize the covered portion of the demand, the optimal value for κ is $1+\sqrt{3}$ which gives a routing for approximately 0.0670Δ of the demand.

We also consider a version of the rooted clustering problem that is geared for our application to MEDP. First, we allow *partial clustering* where the full demand d_i from a terminal need not be assigned. (One may even allow *splittable clustering* where its demand may be assigned to different clusters, although we do not need this version.) Second, we consider instances where terminals come in pairs s_i, t_i, each with a value d_i. (Wlog, the terminals are all distinct.) For each i, we say that s_i, t_i are *siblings*. A clustering *captures* $x \leq d_i$ of pair i's demand, if at least x demand from each of s_i, t_i is assigned to some cluster (not necessarily the same one). Specifically we prove:

Theorem 2.3 (Paired Cluster Capture). *For any $\kappa \geq 2$, and any edge-normalized instance, there is an edge-disjoint rooted partial clustering satisfying at least $\frac{\kappa-1}{\kappa(\kappa+2)}\Delta$ of the total demand. If terminals come in pairs, then we can capture at least $\frac{\kappa-2}{2\kappa(\kappa+2)}\Delta$ of the pairwise demand.*

In the paired case, the κ value that maximizes the routed demand is $2\sqrt{2}+2$ which gives a routing for approximately 0.0429Δ of the demand.

We now turn attention to proving these results. Our first step is to show that we may work in node-normalized instances.

2.2. Reducing to Node-Normalized Instances.

We make a trivial but important observation. Namely that we may work in node-capacitated instances and hence we may employ concepts from the theory of uniform-capacity confluent flows. In a sense then, rooted clustering lies "between" unsplittable and uniform-capacity confluent flows.

If our instance is undirected, then we first solve the flow from terminals in a standard bidirected version of G, i.e., each edge uv becomes two unit capacity edges $(u, v), (v, u)$. In all cases, we can assume that any node v has been split into v^-, v^+ so that any flow destined to t through v traverses a *node arc* of the form (v^-, v^+); the node arc has infinite capacity. We also place the demand d_v at node v on the node v^-. We solve our standard flow in this graph, and assume wlog that it has acyclic support. If the total load on every node arc is at most 1, then the instance is node-normalized already. Otherwise, if (v^-, v^+) has load more than one, we make multiple copies of the node arc and modify the flow so that each of them has load at most 1. (N.B. we will not require this reduction to preserve planarity.)

We modify the flow as follows. Let the in-neighbors of v in D be $u_1, u_2, \ldots, u_p$. Let d_v be the demand of node v. We add new arcs of the form (v_i^-, v_i^+) as follows. Let f_i denote the original flow on the arc e_i from u_i to v. We first move the demand d_v to v_1^-. We then redirect the flow on $e_1, e_2, \ldots, e_j$ until $d_v + \sum_i f_i > 1$. Let σ be the surplus amount. We then modify each $e_i : i \leq j$ to have head v_1^-, but in addition, we only send $f_j - \sigma$ on the last edge e_j. We then open a new node arc (v_2^-, v_2^+) and make an extra copy of e_j whose head is v_2^-, and carries the surplus flow σ. We repeat this process until we have dealt with all the incoming flows for v. See Fig. 3. There is some danger that an edge-disjoint rooted clustering could lead to a congestion 2 clustering in the original graph. This is because we include two copies of some arcs, such as e_j above. This is not an issue however, since our clusters are based on confluent flow routing. In any such routing, there is at most one edge out of any node (and in particular u_j).

We now work in this node-capacitated instance, and apply ideas from [6] for computing a confluent flow. Note that such a confluent flow actually gives a node disjoint rooted clustering (not just edge-disjoint). Mapping this back to the original graph (be it planar or not) may destroy this, but it yields valid edge-disjoint clusters in the original G.

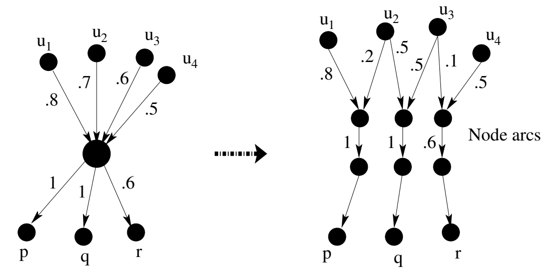

Figure 3. Reducing to node-normalized instances.

2.3. From Confluent Flows to Clusters

Our approach is based on the demand maximization algorithm for confluent flows from [6]; we modify their algorithm and provide an analysis to obtain the Paired Cluster Capture Theorem. In the following, we assume that we are given a node-normalized instance (with possible loss of factor 2 - see Section 2.2.)

Throughout, we let $D = (V, A)$ be the simple input digraph (the undirected case follows from the directed version) and we suppose the instance is node-normalized. In particular, we have a flow vector f that satisfies the routing of some demands s_i, d_i to a single sink $t \in V$, with a maximum flow (i.e., load) of 1 out of each node. A *confluent flow* of the demands to t is a network flow that uses at most one arc out of any node. Thus the support of the flow is just an arborescence rooted at t. Let the in-neighbhours of t be $\{c_1, c_2, \ldots, c_k\}$. Clearly the maximum load at any node occurs at one of the c_i's. Hence, as is done in [6], we actually ignore the sink t, and consider only the c_i's. These are called the *sinks* since wlog we may assume our starting flow f has an acyclic support and also that there are no arcs between a c_i and c_j (since we only require flow to reach one of the sinks). As the algorithm runs, we let b denote the vector of node loads of the sinks; to start, each $b_i \leq 1$ but in time, some of these values may become quite large.

The algorithm [6] repeatedly performs three operations: *node aggregation*, *sawtooth cycle breaking*, and *pivoting*. These operations allow the algorithm to gradually contract "marked" arcs until the only nodes remaining are the sinks c_i. Reversing the contractions then reveals a collection of node disjoint arborescences rooted at the c_i's. They show that if there are still non-sink nodes, then either there is an arc with no flow which can then be deleted, or one of these 3 operations is still possible. We now define the operations in detail.

A *frontier node* is a node u that has an *out-neighbhour* which is a sink, i.e., there is an arc (u, c_j) for some j. (By our assumptions, no sink is a frontier node.) A *decided node* is a frontier node that has exactly one out-neighbhour. *Node aggregation* is an operation that refers to *marking* and then contracting an arc (u, c_j) from some decided node. It is marked since it will become part of our confluent flow tree.

If there is no node aggregation available, the algorithm looks at the residual digraph $\hat{D}$, obtained from $D - t$, by adding a reverse arc (c_j, u) for every arc of the form (u, c_j) where u is some frontier node. A simple directed cycle of length greater than two in $\hat{D}$ is called a *sawtooth cycle*.

The sawtooth cycle breaking operation goes as follows. Consider some sawtooth cycle S. In D, the sawtooth cycle is a cycle with *forward arcs*, i.e.: arcs of D, and *reverse arcs*, i.e.: arcs of D that are used in the reverse direction. The operation increases flow on reverse arcs and decreases flow on forward arcs until at least one of the arcs has its flow decreased to 0; that arc may then be eliminated from $\hat{D}$. (See pseudo-code below.) Note that a reverse arc (c_j, u) is always from a sink to a frontier node and so any reverse subpath of S has length 1 exactly (due to contractions, this edge may actually correspond to an induced path P if we unshrink marked edges). It follows that the operation does not increase the load of any node (except implicitly any internal nodes of P).

```
BREAKSAWTOOTH($D$, $S$, $f$)
$f_{min} = \min\{f(e) : e \in S\}$
For all forward arcs $e$ of $S$
    $f(e) \leftarrow f(e) - f_{min}$
For all reverse arcs $e$ of $S$
    $f(e) \leftarrow f(e) + f_{min}$
```

Finally we consider pivot operations. A *remote node* is some sink c_j that has only one in-neighbhour u, called a *pivot node*, amongst the frontier nodes, but u also has at least one other sink out-neighbhour c_i. If there are no node aggregations or sawtooth cycles, one can show there exists a remote node.

The *pivot operation* takes a pivot node with two out-neighbhours c_i, c_j where the latter is remote. It then shunts the flow on these two out-arcs, one way or the other. For our implementation, we provide a *threshold* parameter κo to decide which way to shunt.

```
PIVOT($D$, $u$, $b$, $f$)
If $b_j - f(u, c_j) \leq \kappa$
    Remove $(u, c_i)$
    $f(u, c_j) \leftarrow f(u, c_j) + f(u, c_i)$
Else
    Remove $(u, c_j)$
    $f(u, c_i) \leftarrow f(u, c_i) + f(u, c_j)$
    Deactivate sink $c_j$
```

Note that pivoting is the only operation that increases the load at some sink.

As long as the load of remote sink c_j is "small", then it will not be deactivated. However, if its load goes above $\kappa + 1$, then the flow is shunted elsewhere and c_j can be shut down or *deactivated*. The reason that $\kappa + 1$ is an upper bound on the cutoff for deactivation is because $f(u, c_j) \leq 1$, and hence if $b_j > \kappa + 1$, then $b_j - f(u, c_j) > \kappa$. It follows that a remote node c_j can either be deactivated, or still has load at most $\kappa + 1$.

In [6], for sink deactivation they use $\kappa = 1/2$. For our application to clustering, we need to consider κ larger than 1 with the optimal being $1 + \sqrt{3}$. We believe that other settings of κ may be useful in other contexts (e.g., setting $\kappa = -1$, where all pivots deactivate, appears a promising version for the (open) problem of confluent routing in $O(1)$ rounds).

Overall, the algorithm runs in polytime, since at every step a node or an arc is removed. It terminates when only sinks remain. The marked arcs then correspond to trees along which flow is routed confluently in the graph D.

After running the algorithm, we say that a tree is *big* if it has total demand greater than $\kappa + 2$. The key idea is to show that by removing a small amount of the demands, we no longer have any big trees. We can then use the resulting trees to act as our clusters.

2.4. Analysis

We now present the proof of Theorem 2.3. The proof for Theorem 2.2 uses similar arguments and is not presented here.

Proof of Theorem 2.3: We call a pivot operation *bad* if it sends flow into a sink whose current congestion was greater than $\kappa + 2$. Note that the sink's congestion could later decrease. Let A be the total amount of bad pivoting which is done by the algorithm. When we perform a pivot step, if a sink c_j is not deactivated, then it had congestion at most $\kappa + 1$ (since node congestion at each non-sink node remains at most 1 throughout the algorithm). After pivoting the flow from c_i into c_j, c_j thus has congestion at most $\kappa + 2$, i.e., the pivoted flow is not bad so A does not increase.

Thus A increases only when a sink c_j is deactivated and then it increases because of flow routed to c_i. This increase is at most 1 since $f(u, c_j)$ is a most 1. If ν is the number of deactivated sinks in the algorithm execution, then we clearly have $A \leq \nu$. Note that any *deactivated tree* (i.e., one obtained by unshrinking the marked arcs leading into a deactivated sink) has total demand greater than κ, and so $\kappa\nu < \Delta$ and thus $A < \Delta/\kappa$.

Now consider the final solution, and any deactivated tree whose sink had load more than $\kappa + 2$. Consider throwing away demands from this tree to bring it down to a maximum load of $\kappa + 2$. If we do this for all trees, the total demand lost is at most Δ/κ of the original demand. Routing the remaining demand now results in node congestion at most $\kappa + 2$ in each arborescence. Hence scaling down the demands by a factor $1/(\kappa+2)$ we route $\frac{\kappa-1}{\kappa(\kappa+2)}\Delta$ demand with node congestion 1.

If demands come in pairs, then eliminating a bad demand may implicitly eliminate demand from its sibling. (We are assuming Δ accounts for the d_i from each sibling.) Hence, the overall loss of pairwise demand may be up to $2\Delta/\kappa$. We are thus guaranteed a total flow of at least $\frac{\kappa-2}{\kappa}\Delta$ from terminals, such that any pair of siblings route the same amount. Scaling down these demands by $\kappa + 2$, these become flows of total value at least

$$\frac{\kappa - 2}{\kappa(\kappa + 2)}\Delta$$

which are in rooted clusters with congestion 1. Dividing by 2 gives a lower bound on the pairwise demand that is captured. ■

3. The Congestion 3 Algorithm for MEDP

3.1. Overview

We now describe the algorithm PLANE-EDP4 ([5]) as well as one of our enhancements using rooted clusters to obtain a congestion 3 algorithm for MEDP.

MEDP can be formulated as follows. We are given a planar graph $G = (V, E)$ and a multiset of node pairs $\mathcal{D} = \{s_1t_1, s_2t_2, \ldots, s_kt_k\}$. Pairs in $\mathcal{D}$ are also called *demands* or *demand edges/pairs*. In the general form, each edge of G also has an integer capacity $u(e)$. A subset (multiset possibly) S of demands is *routable* if there exists a set of paths in G such that there is 1-1 mapping from each pair in S to one of the paths, and each edge lies in at most $u(e)$ of these paths. The objective of MEDP is to find a maximum size routable set. By hanging pendant leaves of unit (or infinite!) capacity, one may assume the demands form a matching. This clearly does not change the optimal solution. Hence we define $X = \{s_1, s_2, \ldots, s_k, t_1, t_2, \ldots, t_k\}$ to be the set of *terminals*, and for each $v \in X$, we denote the (unique) other end of its demand edge by $\sigma(v)$. We refer to $\sigma(v)$ as the *sibling* of v. In a moment, we see that one may reduce (in polytime) the problem to the unit capacity (edge-disjoint) case; the paper focuses on that case.

We consider the natural LP relaxation of MEDP (1). The flow sent on a path P is denoted by $f(P)$. The variable x_i is the total amount of flow that is going to be sent from s_i to t_i. We sometimes refer to $\sum_i x_i$ as the *profit* of the LP solution. For $v \in \{s_i, t_i\}$, let f_v be the total flow that is routed for this pair, i.e.: $f_v = x_i = f_{\sigma(v)}$. f_v is also be called the *demand* of node v. It is well known that this problem can be solved in polynomial time. We let OPT denote the optimal value of the LP; the optimal value for MEDP is clearly less than or equal to OPT.

By polyhedral facts (see Propositions 2.1, 2.2 of [10]) one may assume that the maximum capacity is polynomially bounded, hence we always assume the unit capacity (edge-disjoint) case.

This LP relaxes a feasible solution in two ways. First, we allow fractional x_i whereas in the MEDP problem a pair has $x_i \in \{0, 1\}$. Second, we allow the demand for a terminal pair to be routed along more than one path instead of just one for the MEDP problem, i.e., in MEDP, f is a 0-1 vector. Our goal is to turn any fractional solution to LP, into a solution for MEDP that is within a constant fraction of OPT, at the cost of some edge congestion.

As in [5], we apply a polynomial-time preprocessing phase to reduce the original graph to one where the node degrees are upper bounded by 4. If G was Eulerian and/or planar, these properties may also be preserved by the transformation. The procedure is detailed in [8] and [3]. Thus, we suppose, without loss of generality, that every node has degree at most 4.

After this, the algorithm obtains a "sparse" cut associated with a contour $\mathcal{C}$. The cut is induced by nodes on $\mathcal{C}$ and all of those nodes which lie in the (planar) interior. Two subgraphs are then examined. One is the graph inside $\mathcal{C}$, call it G_1. And the other is the planar graph outside G_1, call it G_2. By sparsity (and bounded degree), we can charge the lost flow (across the cut between G_1, G_2) to G_1, as long as we can capture a constant fraction of demands in G_1. We then recurse on G_2.

The heart of the algorithm is thus to find a constant-factor, constant-congestion algorithm in G_1. Using the so-called 1-Cut reduction on G_1, we reduce to the "hard case". This is when we have a 2-node-connected graph G_C, whose outside face is a simple cycle C. This graph satisfies the following important property. Let f_v denote the total flow terminating at v on flow paths entirely in G_C. Then there is a single "sink" flow that routes f_v/M^* to C for each v (for some choice of constant M^*).

At this point, PLANE-EDP4 uses a 2-phase algorithm to obtain a large routable set in G_C. We describe this part in detail now.

3.2. Phase 1 clustering

Let $p = \sum_v f_v$ be the total fractional flow of the LP solution on flow paths contained in G_C. If $p = O(1)$, it suffices to route one demand pair to get a constant fraction of the optimal solution, so we can suppose that $p > M$ for some sufficiently large M (different from M^* above).

In Phase 1 of the PLANE-EDP4 algorithm, they build connected clusters $R_1, R_2, \ldots R_h$ with the following properties. (i) Each demand is assigned to exactly one cluster and the total demand assigned to R_i is $\Theta(1)$; (ii) each edge lies in at most two R_i's and (iii) the clusters contain distinct nodes $m_i \in C$ which act as "boundary representative" for each cluster. Note that it could be the case that lots of clusters already contain sibling pairs. If so, we can always get a constant fraction of p by simply routing one pair in each such cluster. Thus it is assumed that this is not the case, and that any remaining demand is between terminals assigned to distinct clusters.

We now show how to modify their procedure to obtain edge-disjoint clusters; this will result in a congestion 1 routing for Phase 1. We do this by applying our rooted clustering techniques (namely, the Paired Cluster Capture Theorem 2.3) to the (edge-normalized) routing of $(f_v/M^* : v \in V)$ to C. This produces edge-disjoint clusters that capture a constant total weight of demand pairs. I.e., $\Omega(p)$ of our demand pair flows is captured.

3.2.1. C-Truncation Procedure: It turns out, however, that for our new Phase 2 routing, we also need that each cluster contains a *unique* node from C. (This is important for technical reasons later.) The Paired Cluster Capture method will satisfy this extra constraint as long as our original flows to C sent at most one unit of flow through each node of C (i.e., any flow arriving at a node of C should terminate there).

In *C-truncation*, we consider each demand ab one by one. Consider the flow paths to C from a, b respectively. If any of these flow paths uses more than one node of C, we truncate it at the first such node v. Thus we think of this flow now entering the root (i.e., outside face C) at the node v instead of its original destination. The only problem is when v has already been the destination for a full unit worth of other flow paths; we then say v is *saturated*. Whenever we get to a point where a node v is saturated, then we drop the flow paths from ab that we are trying to re-route to v. We charge any such lost flow for ab to the previous demands which used v. Since each node v has degree at most 4, the total charging that involves flows through v is at most 3. Hence there is a natural charging scheme where each surviving demand flow gets charged at most 6 times by later demands which are "blocked" by one of its flow paths to C.

Hence, we ultimately build clusters such that for each node v, we can denote by $m(v)$ the unique node of C in v's cluster. (As before, we can assume the case where any siblings lie in distinct clusters.) We can now proceed with these new clusters just as in PLANE-EDP4.

3.2.2. The Fractional OS Instance: To understand Phase 1 routing, we must say something about the demands for the Phase 2 routing. These arise from a fractional Okamura-Seymour (OS) instance (all demands have endpoints on C) which is created as follows.

For each terminal v with sibling $v' = \sigma(v)$. We create a demand edge between $m(v)$ and $m(v')$ of weight f_v (where we now use the values as modified by the rooted clustering algorithm from the previous section). In [5], it is shown that such a fractional collection of demands (scaled down further by an appropriate constant) satisfies the cut condition in G_C. (Recall that C is a simple cycle, since we are now looking at a 2-node-connected instance G_C.) Let us denote the resulting demands by a *demand graph* H, where for each $f \in E(H)$, we let $d_f \in [0,1]$ denote the demand value. Since both ends of any demand edge lie on the face C, the OS Theorem (see [15]) states that for any such instance, if the d_f's are integral, the cut condition holds and $G_C + H$ is Eulerian, then the set of demands is (integrally) routable.

We cannot yet apply the Okamura-Seymour Theorem since H has fractional demands. Instead they construct a subset of size $\Omega(p)$ of H's edges, which also satisfy the cut condition with demand weights 1; we describe this next.

3.2.3. From Fractional to Integral OS Instances: PLANE-EDP4 selects a subset of H's demands via an *abstract capacity ring lemma* (Theorem 3.5 [5]) as follows. As noted, H's demands obey the cut condition in G_C. They view these demands as occurring on an "abstract ring" identified with C. For each pair of edges e_i, e_j on C, there is an associated integer capacity $\mu(ij)$; this value is just the minimum capacity of a cut in G_C that contains precisely two edges of C: e_i and e_j.

The ring lemma shows how to convert the fractional demands of H into $\Omega(p)$ integral demands on the ring which almost obey the capacities; moreover, if the capacities $\mu(ij)$ come from a 2-connected graph (which we assume G_C is at this point), the chosen integral demands actually do obey the cut condition. The selected set of demands is denoted by a demand graph H', and we will refer to them as the *OS-selection* demands. (We mention that the ring lemma does not require clusters to have $\Theta(1)$ demand, but only $O(1)$ demand - this is all we guarantee in present form of Theorem 2.3.)

The OS-selection demands have the property that at most one terminal v from any R_i is involved. So if $st \in H'$, then there is an associated $f(s), f(t)$ in distinct clusters such that $s = m(f(s))$ and $t = m(f(t))$. We say that $f(s), f(t)$ are the *associated terminals* with this demand. Phase 1 routing refers to each associated terminal v for the OS-selection demands, (at most one per cluster) using their cluster's capacity to route from v to $m(v)$. This incurs a congestion 2 in the old algorithm, but only congestion 1 in the new clustering.

3.3. *Phase 2 and the Overall Routing*

The overall routing in PLANE-EDP4 is constructed as follows. The path between some pair a, b will use the path from a to $m(a)$ in a's cluster, followed by some path from $m(a)$ to $m(b)$ (this is the Phase 2 routing), and finish by following the path from $m(b)$ to b using b's cluster. Thus with the new clustering, we have immediately reduced the congestion by 1, and hence the overall routing has congestion 3 (using the same Phase 2 routing from [5]).

In Phase 2 of PLANE-EDP4, a congestion 2 routing of the OS-selection demands is achieved as follows. Note that the OS-selection demands obey the cut condition, but we also need the Eulerian condition. To do this, let T be the odd-degree nodes in $G_C + H'$. Since G_C is connected and $|T|$ is even, G_C contains a T-join J, i.e., $J \subseteq E(G_C)$ such that $G_C + H' + J$ is Eulerian. We can now apply the Okamura-Seymour Theorem to obtain a routing for H' in $G_C + J$. This together with the Phase 1 Routing (with our modified Phase 1 clustering) now gives the desired congestion 3 result. We mention that congestion 3 was already proved in an earlier manuscript of the authors, and appears in the thesis [17].

In order to obtain the congestion 2 algorithm, the clean edge-disjoint clusters produced above must share their capacity with the Phase 2 routing. This is our next task.

4. A CONGESTION 2 ALGORITHM FOR MEDP

We now describe our second enhancement to the algorithm PLANE-EDP4 ([5]) to obtain a congestion 2 algorithm.

4.1. *Reducing Congestion Across two Phases*

The high level plan is as follows. Consider Z the set of original associated terminals involved with the OS-selection demands in H'. For each $v \in Z$, let P_v be the path from v to $m(v)$ through its cluster. We refer to these as *stub paths*. We have (by C-truncation) that each $|P_v \cap V(C)| = 1$. We have that H' obeys the cut condition within G_C, but to apply Okamura-Seymour, we need a T-join J so that $G_C + H' + J$ has all even degrees (recall that T are the odd-degree nodes of $G_C + H'$).

Ideally, we could choose the OS-selection demands (i.e., H') so that we do not need any stub paths to create our T-join J (since we need the P_v's for Phase 1 routing). This does not appear possible, but instead we attempt to give some of the stub paths capacity to G_C so it can find a T-join. We need to show that this can be done without sacrificing too many of the demands/terminals Z. Let $P_1, P_2, \ldots, P_{2|E(H')|}$ be the stub paths for $Z = \{v_1, v_2, \ldots, v_q\}$, and let G^* be the graph obtained by deleting their edges from G_C. We refer to $(G_C, H', P_1, \ldots, P_{2|E(H')|})$ as an *augmented OS instance*. In the end we can show

Theorem 4.1. *Let $(G_C, H', P_1, \ldots, P_{2|E(H')|})$ be an augmented OS instance. Let T be the odd-degree nodes of $G_C + H'$ and $G^* = G_C \setminus \cup_i P_i$. Then there exists $\Omega(|E(H')|)$ demands E' with associated terminals $Z' \subseteq Z$ such that resetting $G^* := G^* \cup (\cup_{v \notin Z'} E(P_v))$ now includes a T-join J.*

Hence we keep the stub paths for terminals in Z' so they may route to our selected subset of demands in H' (Phase 1 routing), and use J to achieve Phase 2 routing. (In fact, during this process we alter the T-set to some T'-set, so a little bit of care is needed; we discuss this later.)

4.2. *When all connected components of G^* are T-even*

Let $C_1, C_2, \ldots C_r$ be the connected components of G^*. Since the paths all terminate at the cycle C, and are assumed to only contain one node of C, we have that $V(C)$ is included in one of these components; let us call it C_1 and we refer to this as the *C-component*. This implies that each P_i starts at some component $C_{j(i)}$ and terminates at C_1. We now show

Lemma 4.1. *For an augmented OS instance, if the connected components $C_1, \ldots, C_r$ of G^* are all T-even, then G^* contains a T-join.*

Proof: We use the following basic fact about joins in a graph. For any connected graph, and even subset X of its nodes, the set of edges of the graph contains an X-join. Hence, a first key observation is that if none of the C_i's is *T-odd* (i.e., $|T \cap C_i|$ is odd), then we can find a T-join J in G^*. Since if $G_i = G[C_i]$ is a component, then since $T_i = T \cap V(G_i)$ is even, G_i includes T_i-join; the union of these joins is a T-join J. ■

Hence we could route the OS-selection demands in $G_C + J$ by the Okamura-Seymour Theorem, since $G_C + H' + J$ is Eulerian. Hence, we assume this is not the case and so our objective becomes to fix the T-odd components.

4.3. *Sacrifices*

4.3.1. Overview: The general strategy is to merge any such odd components with C_1. We do this by *sacrificing* some of the P_i's. By sacrificing P_i, we mean that we remove the demand $i\sigma(i)$ from our OS-selection set $E(H')$. If this demand is no longer present, then we no longer need its stub path P_i for Phase 1 routing. Hence the edges of P_i can be included in G^*. Since P_i has one end in C_1, and the other end in $C_{j(i)}$, then in G^* this has the effect of merging these two components (and possibly others involving internal nodes of P_i). We use this approach so as to merge one-by-one any T-odd components in G^* with the C-component C_1. Obviously we must do this in such a way as to protect enough demands in H' from being sacrificed.

4.3.2. Structure of T-odd components: Let us first analyze the structure of a T-odd component C_i, $i > 1$ (we won't be concerned with C_1). By definition of T, $|\delta_{G_C + H'}(C_i)|$ is odd. Since $C_i \cap C = \emptyset$ this means $|\delta_{G_C}(C_i)|$ is odd. (From now on δ will refer to δ_{G_C}.) Since any path P_v with $v \notin C_i$ contributes an even number of edges to this latter cut (it starts and ends outside C_i), we have that there are an odd number of P_v's with $v \in C_i$. We call C_i *big* if this number is at least 3. Otherwise it is *small* and we have a unique path $P_{v(i)}$ starting in C_i, and an even number of paths which may traverse it, each using an even number of edges from $\delta(C_i)$. If $P_{v(i)}$ is the only path intersecting $\delta(C_i)$ for a small component $C_i \neq C_1$, then we call C_i *isolationist*. We do not like these. Note that the component C_i is cut off from $G^* - C_i$ only by the edges in the path $P_{v(i)}$. In the following, we proceed as though there are no such components; we can show how to pre-process the instance to achieve the following lemma (the proof is omitted here).

Lemma 4.2. *An augmented OS instance can always be processed so as to eliminate all isolationist components.*

4.3.3. Selecting candidates for sacrifice: Greedily consider demands $ab \in H'$. We process a, b separately according to the components, $C(a), C(b)$ containing them. (Recall that we now have $C(a) \neq C(b)$.) If $C(a)$ is T-even or contains the outside face C we do nothing. Otherwise $C(a)$ is T-odd and is disjoint from $V(C)$. We try to *protect* a's path P_a which starts in $C(a)$ and ends in the C-component C_1. If $C(a)$ is big, then we pick some path $P_z \notin \{P_a, P_b\}$ which also starts in $C(a)$. P_z is *selected for sacrifice* in the sense that $z\sigma(z)$ is slated for possible removal from H' if we ultimately keep ab. We also say this demand $z\sigma(z)$ *protects the component* for a. Note that if $z\sigma(z)$ is ultimately *sacrificed*, i.e., removed from H', then its path P_z is freed up, and this has the effect of merging $C(a)$ with the component C_1 in G^* (where G^* now contains the edges from P_z). If this does occur, then we would say that $C(a)$ is *merged*.

Now let us consider the case $C(a)$ is small. Here we choose some P_z with $z \notin C(a)$ that passes through $C(a)$. (Such a path exists since we have no isolationist components.) We try to choose z such that $z \neq b$. If this is possible, then we protect $a/C(a)$ and select $z\sigma(z)$ for sacrifice as before. If this is not possible, then we call the pair (or $C(a)$)) ab *cozy*; it has the property that only P_a, P_b intersect $\delta(C(a))$. Such a demand is not protected. In the end, we have that for every demand ab we either determine that it is cozy, or we protect both $C(a), C(b)$ (by selecting for sacrifice up to two other demands). We let $\mathcal{F}_P$ denote an auxiliary graph whose nodes are the demands, and there is an arc from a demand ab to any other demand that was selected for sacrifice to protect ab. We have that this graph has out-degree at most 2.

4.3.4. Routing cozy demands: Note that either we get a constant fraction of protected demands, or we end up with $\Omega(p)$ cozy demands. In the latter case, we can actually route a lot of cozy demands in one go.

Lemma 4.3. *In an augmented OS instance where there are $\Omega(p)$ cozy demands, it is possible to route all cozy demands simultaneously.*

Proof: Any cozy demand ab has the property that say a lies in a component $C(a)$ containing a single terminal. Moreover, the path P_b routes through $C(a)$ and is the only such stub path doing so. We route a, b by merging P_b with an appropriate path in $C(a)$. Hence the cozy demands are simultaneously routable. ■

So we assume from now on that all cozy demands have been thrown out.

4.3.5. Sacrificing demands: Let us now examine $\mathcal{F}_P$. Delete any unprotected nodes (i.e., demands) in this graph. In what remains there are still $q = \Omega(p)$ protected nodes, each of out-degree at most 2. Hence $\mathcal{F}_P$ has at most $2q$ edges. It follows by standard arguments that there is a stable set of size $\Omega(q)$. Find a subgraph of size $\Omega(q)$ and maximum degree $O(1)$; now greedily pick a stable set S. The demands not in the stable set S are sacrificed.

Now let G^* be the graph obtained from G_C by deleting the edges of paths P_a corresponding to any demand in S (or adding to old G^*, stub paths for demands not in S). We must have that in this graph every component is T-even. If there was a T-odd component, then there would be 2 and hence some such component C' that is disjoint from C_1 (the one including C). This component must include one of the original C_i components, $i > 1$. This means there is some path P_v with $v \in C_i$ which was not sacrificed (in particular, $v\sigma(v)$ is not cozy). But then it should correspond to a demand in S, which is impossible since every demand in S was protected by at least one (and up to two) demand that was sacrificed, i.e., some such path P_w has all its edges in G^*; this would merge C_i with C_1, a contradiction.

This establishes

Lemma 4.4. *Let $(G_C, H', P_1, \ldots, P_{2|E(H')|})$ be an augmented OS instance with neither isolationist nor cozy components. Let S be a stable set in the auxiliary graph $\mathcal{F}_P$ and let G^* be the graph obtained from G_C by deleting the edges of paths P_a corresponding to any* *demand selected in S. Then, all connected components in G^* are T-even.*

Thus there are no T-odd components in G^*, and so it contains a T-join. There is one last complication. After sacrificing so many demands, we are now trying to route a new demand graph H'', consisting of the protected demands in S only. So the odd-degree nodes in G_C+H'' is some new set $T' \neq T$. However, the difference is only on nodes in the outside face C, and these all lie in a common component of G^*. Hence every component of G^* must also be T'-even (as well as T-even). Thus G^* contains a T'-join, and so by Okamura-Seymour there are edge-disjoint paths in $G_C + G^*$ that satisfy the demands in H''. That is, we have a Phase 2 routing with congestion 1 in G_C+G^*. This can now be merged with the Phase 1 routing which only uses the edges of stub paths P_v that were protected, i.e., they did not use edges in G^*.

5. Conclusions

A very interesting question is to develop methods to get a constant factor approximation for the weighted MEDP problem in planar graphs. Another direction is to strengthen the natural LP relaxation to obtain a constant integrality gap for all planar graphs (including the grid-like structures). In a similar vein, it appears that the techniques from Section 4 together with an additional technical result, could give an $O(1)$-approximation for Eulerian planar instances. This is slightly less natural from the approximation perspective, but it gives a strictly stronger result. We hope to include this in a journal version.

References

[1] M. Andrews, "Approximation algorithms for the edge-disjoint paths problem via Räcke decompositions," in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, 2010, pp. 277–286.

[2] S. Antonakopoulos, C. Chekuri, B. Shepherd, and L. Zhang, "Buy-at-bulk network design with protection," *Foundations of Computer Science, 2007. FOCS '07. 48th Annual IEEE Symposium on*, pp. 634–644, 2007.

[3] C. Chekuri, S. Khanna, and F. B. Shepherd, "Edge-disjoint paths in planar graphs," in *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, 2004, pp. 71–80.

[4] ——, "The all-or-nothing multicommodity flow problem," in *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2004, pp. 156–165.

[5] ——, "Edge-disjoint paths in planar graphs with constant congestion," in *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2006, pp. 757–766.

[6] J. Chen, R. D. Kleinberg, L. Lovász, R. Rajaraman, R. Sundaram, and A. Vetta, "(Almost) tight bounds and existence theorems for single-commodity confluent flows," *J. ACM*, vol. 54, no. 4, p. 16, 2007.

[7] Y. Dinitz, N. Garg, and M. X. Goemans, "On the single-source unsplittable flow problem," *Combinatorica*, vol. 19, no. 1, pp. 17–41, 01 1999/01/04/. [Online]. Available: http://dx.doi.org/10.1007/s004930050043

[8] A. Frank, "Packing paths, cuts, and circuits - a survey," in *Paths, Flows and VLSI-Layout*, B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, Eds. Springer Verlag, 1990, pp. 49–100.

[9] N. Garg, V. V. Vazirani, and M. Yannakakis, "Primal-dual approximation algorithms for integral flow and multicut in trees." *Algorithmica*, vol. 18, no. 1, pp. 3–20, 1997.

[10] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis, "Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems* 1," *Journal of Computer and System Sciences*, vol. 67, no. 3, pp. 473–496, 2003.

[11] R. Hassin, R. Ravi, and F. S. Salman, "Approximation algorithms for a capacitated network design problem," *Algorithmica*, vol. 38, no. 3, pp. 417–431, 03 2004/03/01/. [Online]. Available: http://dx.doi.org/10.1007/s00453-003-1069-7

[12] K.-i. Kawarabayashi and Y. Kobayashi, "The edge disjoint paths problem in eulerian graphs and 4-edge-connected graphs," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '10. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010, pp. 345–353. [Online]. Available: http://portal.acm.org/citation.cfm?id=1873601.1873630

[13] J. Kleinberg and É. Tardos, "Approximations for the disjoint paths problem in high-diameter planar networks," in *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*. ACM, 1995, pp. 26–35.

[14] J. M. Kleinberg, "An approximation algorithm for the disjoint paths problem in even-degree planar graphs," in *FOCS '05: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2005, pp. 627–636.

[15] H. Okamura and P. D. Seymour, "Multicommodity flows in planar graphs," *Journal of Combinatorial Theory, Series B*, vol. 31, no. 1, pp. 75–81, 1981/8. [Online]. Available: http://www.sciencedirect.com/science/article/B6WHT-4KBW025-8/2/9b4489ece0a97e9d8340d69948600501

[16] P. Raghavan and C. Thompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, pp. 365–374, 1987.

[17] L. Séguin-Charbonneau, "Confluent, bifurcated and unsplittable flows," Master's thesis, McGill University, Montréal, 2009.