Mathemati cal Programming

FULL LENGTH PAPER

Series A

Maximum edge-disjoint paths in planar graphs with congestion 2

Loïc Séguin-Charbonneau[1] · F. Bruce Shepherd[2]

Received: 30 November 2016 / Accepted: 24 April 2020 / Published online: 20 May 2020

Abstract
We study the maximum edge-disjoint path problem (MEDP) in planar graphs $G = (V, E)$ with edge capacities $u(e)$. We are given a set of terminal pairs $s_i t_i$, $i = 1, 2 \dots, k$ and wish to find a maximum *routable* subset of demands. That is, a subset of demands that can be connected by a family of paths that use each edge at most $u(e)$ times. It is well-known that there is an integrality gap of $\Omega(\sqrt{n})$ for the natural LP relaxation, even in planar graphs (Garg–Vazirani–Yannakakis). We show that if every edge has capacity at least 2, then the integrality gap drops to a constant. This result is tight also in a complexity-theoretic sense: recent results of Chuzhoy–Kim–Nimavat show that it is unlikely that there is any polytime-solvable LP formulation for MEDP which has a constant integrality gap for planar graphs. Along the way, we introduce the concept of *rooted clustering* which we believe is of independent interest.

Keywords Edge-disjoint paths · Integrality gap · Flows

Mathematics Subject Classification 90C27 · 90C05

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 graph $G = (V, E)$ and a set of node pairs (*demands*) $\mathcal{D} = \{s_1t_1, s_2t_2, \dots, s_kt_k\}$. Define $X = \{s_1, s_2, \dots, s_k, t_1, t_2, \dots, t_k\}$. The nodes in X are called *terminals* and the two terminals in a pair are called *siblings*; for $v \in X$ we denote its sibling by $\sigma(v)$. A standard

✉ F. Bruce Shepherd
fbrucesh@cs.ubc.ca

Loïc Séguin-Charbonneau
loicseguin@gmail.com

1 Cégep Édouard Montpetit, Longueuil, QC, Canada

2 Department of Computer Science, University of British Columbia, Vancouver, BC, Canada

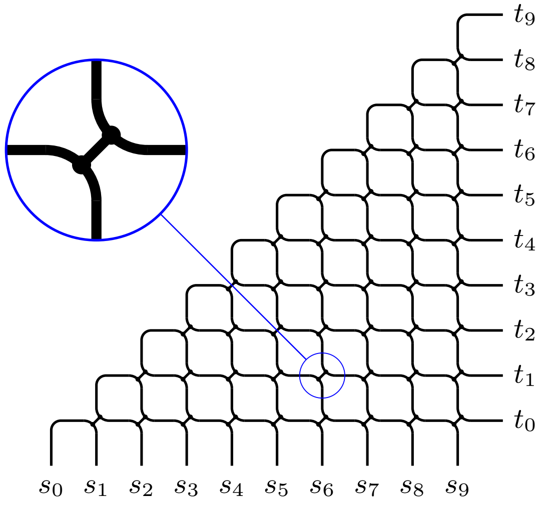

Fig. 1 The $\Omega(\sqrt{n})$ integrality gap instance from [16]

reduction (adding pendant leaves) allows one to assume without loss of generality, that the terminals are distinct. Thus the demands form a matching. 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, the case where each edge e has an integer capacity $u(e)$ is often considered. In this version, we seek a collection of paths such that each edge is contained in at most $u(e)$ paths. Since the methods can handle integer capacities in a straightforward fashion (cf. [3,14]), it is usually convenient to work with unit capacities, i.e., the "true" edge-disjoint setting.

We work with the natural linear programming (LP) formulation for MEDP (see (1) in Sect. 3). Let LP- OPT denotes its optimal value. It is well-known that this LP may have a $\Omega(\sqrt{n})$ integrality gap [16], even in undirected planar graphs. The standard example consists of an instance on a grid, see Fig. 1.

Kleinberg and Tardos [19] thus initiated the study of *low congestion routings*. 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. For instance, Ω(LP- OPT) may be achieved in a general graph with $O(\log n)$-congestion via randomized rounding [22]. This agenda was motivated by the grid example since one may route all demands if edge congestion 2 is allowed. It is natural to ask if one can approximate LP- OPT if we are allowed to use integer solutions corresponding to "low"-congestion routings. More specifically, we are interested in whether one can compute constant-congestion routings of Ω(LP- OPT) demands.

An early positive result in this vein [3] shows that for planar graphs, the LP gap is at most $O(\log n)$ if one allows edge congestion 2. A $O(\log^2 n)$ approximation, with no congestion, was later given for even, planar graphs [20]. Using completely different methods, it was shown [5] 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.

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

Since the conference version of this paper it has been explicitly conjectured [6] that the natural LP has an $O(1)$-approximation with congestion 2 for any fixed minor-closed class. Using results in this paper, they have established congestion 3 routings both in the case of bounded treewidth and bounded genus graphs. It had also been asked whether a tightening of the natural LP relaxation could actually give a constant integrality gap for all planar graphs without congestion. Recent results [10] indicate that this is not possible under certain complexity theoretic assumptions. Hence at some level, guaranteeing a solution within a constant factor of the optimal solution to MEDP would appear to require congestion. This is in contrast to the case of maximum all-or-nothing flows where a constant-factor congestion-1 algorithm has been devised [18].

The theory of low-congestion routings has also been developed in general graphs. An initial breakthrough [2] gave a polylogarithmic lower bound, even for the all-or-nothing flow version. The first breakthrough in terms of upper bounds was [1] where a polylogarithmic approximation is given for MEDP using $O(poly \log \log)$ congestion. Since the conference version of the present paper, the picture has been largely completed. In [9], a polylogarithmic approximation with $O(1)$ congestion was found, and this was then strengthened to hold in the congestion 2 setting [8]. Moreover, a polylogarithmic approximation, with congestion 2 is now known for the node-disjoint setting [4]. These papers build on the framework of [3] but diverge from the theory needed for constant-factor approximations and we do not discuss them further.

OVERVIEW OF THE APPROACH

We now give a high level view of the algorithm from [5] which we henceforth refer to as PLANE-EDP4 (more details are given in Sects. 3–5). The algorithm first performs several standard reductions; for instance, it reduces the graph G to have bounded degree and assumes that the demand edges $s_i t_i$ form a matching. PLANE-EDP4 then repeatedly finds a "sparse" cut and deletes the edges in this cut. Each such cut breaks the graph into two planar parts G_1, G_2 where one side, G_1, is chosen to be minimal in a certain sense. The flow lost on the cut edges is "charged" to the flow within G_1. PLANE-EDP4 then applies an algorithm which we call the *one-time routing procedure*; this extracts a "large" set of demands within G_1 which can be integrally-routed with congestion 4. By large, we mean that the number of demands routed is a constant fraction of the fractionally routed demands from LP- OPT which used flow paths that "touched" G_1, i.e., paths entirely within G_1 or crossing the sparse cut. After this, the algorithm recurses on G_2. The meat of the algorithm is thus the *one-time routing procedure* which extracts the demands and routing in G_1. We highlight this now since it forms the key algorithmic requirement for our algorithm—see Theorem 4.1.

The One-Time Routing Algorithm *We are given a fractional flow on paths which are incident to at least one node of G_1. If the total flow routed is F, then there is an algorithm to find a congestion* 2 *integral routing entirely within G_1, which routes $\Omega(F)$ demand pairs.*

In PLANE-EDP4 this one-time routing is based on a two-phase algorithm which converts a fractional routing into an integral one with congestion 4. This consists of a *clustering phase* and a *routing phase*. In [5] a congestion factor of 2 arises in each phase. Using a new clustering scheme, we improve Phase 1 to only incur congestion 1 (Sect. 2). In Sect. 5 we show how to turn this into a congestion 3 algorithm. In order to achieve an overall congestion 2, however, Phase 1 cannot be done independently; it must share some capacity with the second phase (Sect. 6). This completes the high level view of PLANE-EDP4 and our intended modifications. We now discuss more details from Phase 1 as it motivates the rooted clustering problem introduced in the next section.

The clustering phase of PLANE-EDP4 starts from a fractional multi-flow with the property that every positive flow path intersects an outside face C of a given planar graph. Let X denote the terminals which are the endpoints of these flow paths. If $v \in X$, we let d_v denote the total flow on paths terminating at v. The goal is to find a collection of subgraphs $H_1, H_2, \dots, H_c$ called clusters, and an assignment of terminals to clusters. Let X_i denote the terminals assigned to H_i. These must satisfy the following.

1. each edge occurs in at most 2 clusters
2. every terminal of H_i is connected to C by a subpath of H_i
3. for each H_i, $\sum_{v \in X_i} d_v = O(1)$.

Ideally we would strengthen the first condition so that the clusters are edge-disjoint. Unfortunately, such edge-disjoint clusterings may not exist in directed graphs (Fig. 2). Instead we sacrifice some of our terminals. We show that by refocusing on a subset of demand pairs, such an edge-disjoint clustering exists. We abstract this clustering phase to a stand-alone problem called *rooted clustering*. The "root" in this context plays the role of the outside face C to which clusters must be connected (i.e., Condition 2 above). This problem is introduced in Sect. 2.

In Sects. 3–5 we show how our rooted clustering algorithm improves the congestion bound for planar MEDP to 3. Finally, in Sects. 6–6.1 we improve the overall congestion of the MEDP algorithm to 2, thus obtaining Theorem 1.1.

2 Rooted clustering

In this section we introduce rooted clustering, a problem that intuitively lies between single-sink unsplittable flows [12] and single-sink confluent flows [7]. 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* $s_1, \dots, s_k$ each with a weight $d_i \in [0, 1]$. A *rooted clustering* is a collection of subgraphs (called *clusters*) $H_1, \dots, H_c$ each containing t, and an assignment $f : \{s_1, s_2, \dots, s_k\} \to \{H_1, H_2, \dots, H_c\}$ with the following properties

1. if $f(s_i) = H_j$, then $s_i \in V(H_j)$
2. for each i, there is a (directed) path from each node in H_i to t
3. $\sum_{s_i \text{ assigned to } H_j} d_i = O(1)$.

Note that a terminal s_i may appear in more than one cluster but it is only assigned to one.[1] The *congestion* of an edge is the number of clusters in which it appears; the clustering has congestion c if each edge has congestion at most c. If the congestion is 1, then we refer to it as an *edge-disjoint* (or congestion 1) rooted clustering.

Rooted clusterings are closely related to their corresponding *single-sink flow problems*. In this problem, the terminals are associated with a demand to route d_i to the node t. We assume that G is directed and t is a sink.[2] A *single-sink flow vector* is then $f : E(G) \to \mathbb{R}_{\geq 0}$ satisfying: $f(\delta^+(s_i)) - f(\delta^-(s_i)) \leq d_i$ for each terminal s_i, and $f(\delta^+(v)) - f(\delta^-(v)) = 0$ for each non-terminal $v \neq t$. Depending on the context, we may view these as being either node-capacitated or edge-capacitated. We call an instance *edge-normalized* (resp. *node-normalized*) if there is a flow such that $f(e) \leq 1$ for each $e \in E(G)$ (resp. for any node $v \neq t$ $f(\delta^+(v)) \leq 1$).

Single-sink *unsplittable* flows (demand from s_i must be routed on a single path) have been studied from the perspective of both congestion and throughput (see [12]). Note that the existence of an edge-disjoint rooted clustering implies the existence of an (unsplittable) routing of the demands to t, with "edge congestion" at most the maximum total demand in a single cluster. In context of unsplittable flows, one usually assumes that the cut condition (2) is satisfied. By the Max-flow Min-Cut Theorem, this condition is equivalent (for unit capacity graphs) to asserting that an instance is edge-normalized. We use the same natural necessary condition for clustering.

The clustering phase of PLANE-EDP4 produces the following for rooted clustering.[3]

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

Unfortunately, an edge-disjoint rooted clustering need not exist for directed graphs (we do not know the answer for undirected). This is shown by the example in Fig. 2 where the bottom layer nodes are all adjacent to the root (which is not drawn).[4] Analysis in [7] shows that this instance is node-normalized. Now consider an edge-disjoint rooted clustering. containing the top node. The cluster containing the top node must include a directed path P to the root. Since the edges are all directed downwards, every terminal on this path must actually be contained in the same cluster. 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.

To address the above difficulty, we relax the condition that we have a *total* clustering, i.e., that every demand is assigned to a cluster.

[1] One may relax the condition that a terminal's demand is assigned to only one cluster. A *splittable* rooted clustering is one where a demand of d_i may be split across multiple clusters.

[2] If G is undirected, we bi-direct edges and "trim" t to be a sink.

[3] Their set-up is slightly different (e.g., they only consider bounded degree graphs) consider planar graphs and clusters being but their ideas easily yield this general result.

[4] This is essentially the same as one given in [7] as an example of a $\Omega(\log n)$ gap for the congestion minimization LP for confluent flows.

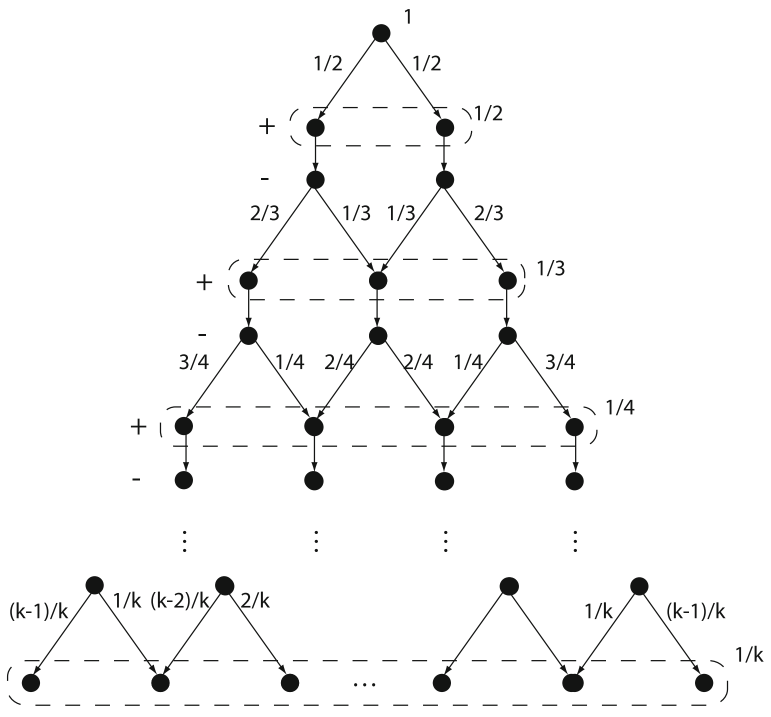

Fig. 2 There is no arc-disjoint rooted total clustering (even if demands can be split across different clusters)

2.1 Partial clustering

A *partial* rooted clustering is one where some demands may not be assigned to a cluster. Our first result shows that we may assign a large fraction of the demands to an edge-disjoint clustering.

Theorem 2.2 *For any edge-normalized rooted clustering instance, there is an edge-disjoint rooted clustering that assigns at least $\frac{\Delta}{3}$ of the total demand $\Delta = \sum_i d_i$.*

We next consider a version of the rooted clustering problem that is geared for our application to MEDP. First, we allow *fractional clustering* where the full demand d_i from a terminal need not be assigned.[5] If $\delta \leq d_i$ of the demand is assigned, then the clustering is said to *satisfy* δ of terminal i's demand. Second, in our instances the terminals come in pairs s_i, t_i, each with a value d_i. (Again, the terminals are assumed to be distinct.) A clustering *captures* ϵ of pair i's demand, if at least ϵ demand from each of s_i, t_i is assigned to some cluster (not necessarily the same one).

Formally, a fractional clustering can be defined as a fractional matching in a bipartite graph where one side of the bipartition consists of terminals, and the other side consists of the clusters. There is an edge between terminal v and cluster H_i if $v \in H_i$. The

[5] One could also consider *splittable clusterings* where a fraction of the demand is assigned across several clusters. We do not need to resort to this version.

fractional matching x should satisfy: (i) at most one edge of $\delta(v)$ is in the support of x (since we assign any terminal to at most one cluster), (ii) for each terminal $v \in \{s_i, t_i\}$, $x(\delta(v)) \leq d_v$, (iii) for each cluster H_j, $x(\delta(H_j)) = O(1)$.[6] We *capture* ϵ from demand i, if $x(\delta(s_i)), x(\delta(t_i)) \geq \epsilon$.

We then prove:

Theorem 2.3 (Fractional Clustering and Capturing Pairs) *For any $\kappa \geq 2$, and any edge-normalized instance, there is an edge-disjoint rooted fractional clustering which satisfies 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 0.0858Δ of the demand.

We now turn attention to proving these results. Our first step is to show how to use the concept of confluent flows.

2.2 Confluent flows and rooted clusters

We have seen how rooted clustering is related to unsplittable single-sink flows (see introduction to this section). We now show that algorithms for *confluent* single-sink flows can be used to produce rooted clusters. As before we assume a directed graph $G = (V, E)$ with sink node t, and terminals s_i each with a demand d_i to route to t. A single-sink flow f is called *confluent* if $|supp(f) \cap \delta^+(v)| \leq 1$ for each $v \in V(G)$. In other words, the support induces a (possibly non-spanning) arborescence into t. (Recall that an *arborescence* rooted at node t is a connected graph where every node has out-degree 1 except for t which is a sink.) In this problem one normally is only given node capacities. Recall the instance is *node-normalized* if it has a flow such that for each node $v \neq t$, its *load* $f(\delta^+(v))$ is at most 1. (Note that any node-normalized instance is also edge-normalized.)

Let the neighbhours of t in D be $\{v_1, v_2, \ldots, v_k\}$. In a confluent flow, it is clear that the maximum load at any node occurs at one of the v_i's. In [7], it is proved that for any node-normalized instance, there is a feasible confluent flow which routes $\frac{1}{3}$ of the total demand. (This is done in an all-or-nothing fashion; each demand is either fully routed or not routed at all.) We use this demand maximization algorithm to obtain a rooted clustering.

To see the connection, consider a feasible confluent flow f which routes some of our demand; let T be its support. Since the load at each v_i is at most 1, the subtree of T rooted at v_i contains at most 1 unit of demand. Hence these subtrees identify node-disjoint (not just edge-disjoint) clusters which are rooted to t. To use this idea, we must show how to turn an edge-normalized instance of rooted clustering into a node-normalized instance for confluent flow. We describe this next.

REDUCTION TO NODE- NORMALIZED INSTANCES. We now describe a routine to convert an edge-normalized (clustering) instance into a node-normalized (confluent flow) instance which has an additional *outgoing edge property* (introduced below). In both

[6] The notation $\delta(X)$ represents the set of edges in the bipartite graph with exactly one endpoint in X.

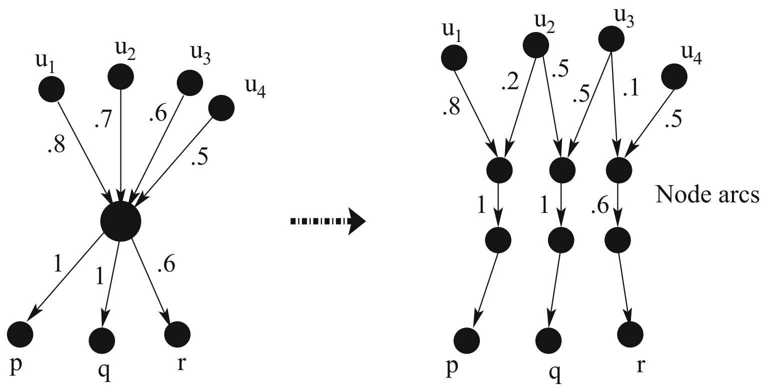

Fig. 3 Reducing to node-normalized instances

directed or undirected rooted clustering instances, we start by running a standard flow algorithm which produces a flow in our initial (edge-capacitated) instance, which can be further assumed acyclic. Moreover, we can then even assume that in this instance, any node $v \neq t$ has been split into two nodes v^-, v^+ so that any flow path through v uses a new *node edge* of the form (v^-, v^+). If the total load on any node edge is at most 1, then the instance is node-normalized already. If this is not the case, then we modify the instance by making multiple copies of node edges for any v which fails this condition.

We process the nodes farthest from t first, i.e., ones appearing earlier in the acyclic order. Hence, when we examine v, we already examined all of its predecessors. First, partition the edges in $\delta^+(v)$ such that for each set X_i in the partition, the total flow on these edges was at most 1. The edges of X_i will now correspond to outgoing edges from a node v_i^+, where $e_i = (v_i^-, v_i^+)$ is a new node edge. This allows our reduced graph to have the *outgoing edge property*: any original edge in $\delta^+(v)$ is the outgoing edge of at most one new node v_i^+. Next we greedily re-route the original flows entering v so that they flow into one of the new nodes v_i^- where e_i still has available capacity. It could be that the flow on an old edge $e = (u, v)$, fills up e_i and any residual from from e must be routed into e_{i+1}. This maintains the outgoing edge property since it creates two copies of edge e in the reduced graph, say (u_j^+, v_i^-) and (u_j^+, v_{i+1}^-). The reduction is depicted in Fig. 3; N.B. we do not require this reduction to preserve planarity.

Theorem 2.2 *For any edge-normalized rooted clustering instance, there is an edge-disjoint rooted clustering that assigns at least $\frac{\Delta}{3}$ of the total demand $\Delta = \sum_i d_i$.*

Proof We seek an edge-disjoint rooted clustering by first applying node-normalization to create an associated instance G'. We then apply the demand maximization routine of [7] which routes one third of the demand $\sum_i d_i$. As discussed above, such a confluent flow gives a node-disjoint rooted clustering in G'. Reversing the node-normalization process may destroy this back in the original graph as it involves collapsing, for each node v, all of its new nodes v_i^+, v_i^- back into v. This yields a valid edge-disjoint clustering in G due to the outgoing edge property in G'.

This proof also yields the following.

Lemma 2.4 *Let G be an edge-normalized instance for rooted clustering and G' be the reduced node-normalized instance with outgoing edge property. If G' has a node-disjoint rooted clustering which assigns all the demands in J, then G has an edge-disjoint rooted clustering assigning demands in J.*

We now show how to adapt the demand maximization algorithm of [7] to produce the paired capture Theorem 2.3.

2.3 Confluent flow demand maximization

We first give an overview of the demand maximization algorithm for confluent flows from [7]; we then modify it and provide an analysis to obtain the Fractional Clustering and Capture Pairs Theorem 2.3.

The algorithm starts with a node-normalized flow (routing all of the demands $\sum_i d_i$) in an acylic graph. Henceforth, they work in $G\backslash t$ and let $v_1, v_2, \ldots, v_k$ be the neighbhours of t. One can assume there are no edges between sinks v_i and v_j since we only require flow to reach t. Hence the v_i's are assumed to be sink nodes. As the algorithm runs, b denotes 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 [7] repeatedly performs three operations: *node aggregation*, *sawtooth cycle breaking*, and *pivoting*. It performs these to simplify the graph by gradually contracting edges until the only nodes remaining are the sinks v_i. We refer to a contracted edge as *marked* since they identify the arborscences rooted at v_i's in the original graph, by reversing the contractions. One key technical result is to show that if there are still non-sink nodes, then either there is an edge with no flow which can then be deleted, or one of the 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 edge (u, v_j) for some j. (By our assumption, 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 the edge (u, v_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 an *auxiliary digraph* $\hat{G}$ related to the standard residual digraph for flows. It is obtained from $G\backslash t$, by adding a reverse edge (v_j, u) for every edge of the form (u, v_j) where u is some frontier node. A simple directed cycle of length greater than two in $\hat{G}$ is called a *sawtooth cycle*.

Consider some sawtooth cycle S. Back in G, the sawtooth cycle is a cycle with *forward edges*, i.e.: edges of G, and *reverse edges*, i.e.: edges of G that are used in the reverse direction. The sawtooth cycle breaking operation is:

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

Note that a reverse edge (v_j, u) is always from a sink to a frontier node. Hence any reverse subpath of S has length exactly 1 exactly. (If we reversed contractions, such a reverse edge may, however, correspond to an induced path P.) It follows that the operation does not increase the load of any node (except implicitly any internal nodes of P). Moreover, the operation guarantees that at least one of the edges will have its flow decreased to 0; that edge may then be eliminated from $\hat{D}$. This operation was introduced in [12] and a theory for them is developed in [13].

Finally we consider pivot operations. A *remote node* is some sink v_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 v_l. If there are no node aggregations or sawtooth cycles, one can show there exists a remote node.

We now define the *pivot operation* which takes a pivot node with two out-neighbhours v_i, v_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 also provide an extra *threshold* parameter κ.

```
PIVOT($G$, $u$, $b$, $f$)
**If** $b_j - f(u, v_j) \leq \kappa$
    Remove $(u, v_i)$
    $f(u, v_j) \leftarrow f(u, v_j) + f(u, v_i)$
**Else**
    Remove $(u, v_j)$
    $f(u, v_i) \leftarrow f(u, v_i) + f(u, v_j)$
    Deactivate sink $v_j$
```

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

The pivot operation can be described in words as follows. As long as a remote sink's load is "small", then any pivot operation that it is involved in, shunts more flow onto it. However, if its load goes above $\kappa + 1$, then the flow is shunted elsewhere and the remote node v_j is shut down or *deactivated*. The reason that $\kappa + 1$ is an upper bound on the cutoff for deactivation is because we have $f(u, v_j) \leq 1$. Hence if $b_j > \kappa + 1$, then $b_j - f(u, v_j) > \kappa$. This ensures that its final load after deactivation is at least κ. Summarizing, any remote node can either be deactivated, or still has load at most $\kappa + 1$.

In [7], 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. For instance, 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 edge is removed. It terminates when only sinks remain. The marked edges then correspond to trees along which flow is routed confluently in the graph G.

2.4 Proof of Theorem 2.3

Proof The proof follows the same lines as for Theorem 2.2. We transform the edge-normalized instance into an associated node-normalized instance of confluent flow with the outgoing edge property (w.r.t original instance). If we create the desired node-disjoint clustering in the new instance, then Lemma 2.4 guarantees the edge-disjoint clusters in the original instance.

Henceforth, our focus is to find the node-disjoint clusters for the reduced node-normalized confluent flow instance. The general idea is as follows. After running the maximum confluent flow algorithm, a tree is called *big* if it has total demand greater than $\kappa + 2$. The key point is to show that by removing a small amount of the demands, we no longer have any big trees. We can thus (almost) use the resulting trees to act as our clusters.

We call a pivot operation *bad* if it increases flow into a sink v_j whose current load b_j was greater than $\kappa + 2$. Note that the sink's load could later decrease. Let A be the sum of all such increases which occur on bad pivots by the algorithm. Since the node congestion of each non-sink node remains at most 1, we focus on sinks in pivot operations. Consider a pivot step on u across sinks v_i, v_j where v_j is remote. Since v_j is not deactivated, it had congestion at most $\kappa + 1$. Hence, if the pivot shunted flow into v_j, its load would be at most $\kappa + 2$, i.e., the pivoted flow is not bad so A does not increase.

Thus A increases only when a sink v_j is deactivated and then it increases because of flow re-routed to v_i. This increase is at most 1 since $f(u, v_j)$ is a most 1. It follows that if ν is the number of deactivated sinks in the algorithm execution, then $A \leq \nu$. Note that any *deactivated tree* (i.e., one obtained by unshrinking the marked edges leading into a deactivated sink) has total demand greater than κ since its corresponding remote node had at least this much load after deactivation. Hence $\kappa\nu < \Delta$ and thus $A < \Delta/\kappa$.

Now consider the final solution and any (now deactivated) subtree whose sink had load more than $\kappa + 2$. Consider reducing demands in this tree to bring it down to a total load of $\kappa + 2$. If we do this for all trees, the total demand lost is at most $A \leq \Delta/\kappa$. We can thus route the remaining $\frac{(k-1)\Delta}{k}$ reduced demand with node congestion at most $\kappa + 2$ in each arborescence. Hence scaling down these 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 reducing demand on say s_i implicitly reduces it from t_i. (Recall, Δ double-counts pairwise demand, i.e., $\Delta = \sum_{v \in X} d_v$ where X is the set of terminals.) Hence, there may be up to an additional amount of Δ/κ which we should reduce on demand from siblings. After this, we have routed at least $\frac{(k-2)\Delta}{k}$ demand with congestion $\kappa + 2$, but each pair of siblings routes the same as one another. We now scale to route

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

demand of paired-up flows with congestion 1. Dividing by 2 gives the lower bound on the pairwise demand that is captured. □

3 MEDP: basic definitions, the LP relaxation and preprocessing

The maximum edge-disjoint path problem (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* or simply *demand pairs*. In the general form, each edge of G has a non-negative 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.

A natural LP relaxation of MEDP is the following:

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

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. The flow sent on a path P is denoted by $f(P)$. The variable x_i is the total amount of flow sent between s_i and t_i. We sometimes refer to $\sum_i x_i$ as the *profit* of the LP solution. It is well-known that this LP can be solved in polynomial time. We let LP- OPT denote the optimal value of the LP; the optimal value for MEDP, MED- OPT, is clearly less than or equal to LP- OPT.

We extend the algorithm PLANE-EDP4 developed in [5] which in turn builds on the work [3]. As in those algorithms, we assume some preprocessing which we outline next.

First, by polyhedral considerations (see Propositions 2.1, 2.2 of [17] for details) one may assume that the maximum capacity is polynomially bounded. Hence we always assume the unit capacity (edge-disjoint) case. Second, we assume a polynomial-time preprocessing phase which reduces 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 [3,14].

Henceforth we assume a unit-capacity instance in a planar graph with maximum degree 4.

3.1 The 1-cut procedure

As in PLANE-EDP4 our algorithm initially computes a flow f via the LP relaxation. The goal is to find a (integral) routing which is within a constant of the fractional optimum $\sum_i x_i$. This is done by repeatedly finding "sparse cuts" to split an instance into two pieces G_1, G_2. We discuss properties of these cuts more formally in Sect. 4.1 but at the high level, sparsity of the cut gives two important properties:

1. it implies that it is sufficient to recurse on G_2 as long as we can route a constant fraction of the flow which is entirely contained in G_1. The lost flow is "charged" to the pairs we route in G_1.
2. the sparse cut routine ensures special connectivity properties within G_1 that allows us to extract a constant fraction of its profit. We call this the *one-time routing* in G_1.

There is one case where (1) is particularly simple and we can directly use a *1-cut reduction* from [7]. We describe this next. Recall that a graph G is 2-node-connected if it is connected and so is $G - v$ for any $v \in V(G)$. A *block* of G is a maximal 2-node-connected subgraph. Any such block is either a single edge, or has the property that every node lies on a cycle (such graphs can be built up by ear decompositions [15]).

We compute the 2-node-connected blocks of G_1. Suppose that half the LP flow lies on paths which traverse more than one block (i.e., using edges from more than one block). Then we use a *Block Tree Routing Algorithm* which routes a number of demands which is at least a constant fraction of the flow entirely within G_1; this is done with congestion 1 in fact. This procedure is described in Theorem 3.10 of [5].

Otherwise half the LP's flow is on flow paths which lie within a single block. We then throw away the other flow and process each individual block. Combining the routings from these blocks yields the desired large routing inside G_1 (to which we charge any flow which is thrown away).

Hence the core technical challenge is to find a large (constant-fraction) routing within each block of G_1. If a block is a single edge, this is trivial. Otherwise it is 2-node-connected and has cycles. It follows that we only need a method for one-time routing in these nontrivial 2-node-connected instances.

4 PLANE-EDP4

The following summarizes the high-level behaviour of PLANE-EDP4 (after preprocessing) as well as our algorithm.

1. Find a maximum solution to the LP (1).
2. Use the current flow to guide a *sparse cut routine* which partitions the nodes of G into G_1, G_2 (in final iteration G_2 is empty).
3. Run the 1-Cut Procedure on G_1. If most of the flow crosses blocks, then we extract a constant-fraction routable set in G_1 using the *Block Tree Routing Algorithm* (Sect. 3.1).

4. Otherwise, extract a constant-fraction (aka large) *one-time routing* in each 2-node-connected block of G_1.
5. Recurse on G_2 using the residual flow.

In PLANE-EDP4 the one-time routing procedure extracts a large edge-disjoint path solution with congestion 4. Our goal is to improve this to congestion 2. This is only possible due to the connectivity properties in G_1 guaranteed by the sparse cutting procedure.

4.1 Properties of the sparse cut routine

We use the sparse cut routine from [5] (Sect. 3.1) which computes a contour $\mathcal{C}$ which only touches nodes of G.[7] This defines G_1 as the subgraph embedded inside of $\mathcal{C}$ and G_2 is the subgraph embedded outside $\mathcal{C}$. (We mention that G_1 could possibly even be disconnected.) If G_2 has no edges, then this is the final iteration.

Two properties are guaranteed by this routine. The first follows from the fact that the contour constructed in Lemma 3.1 [5] is what they call "short".

Property 1 *In G_1, the number of nodes on the contour $\mathcal{C}$ which are incident to an edge outside G_1, is a fraction (say at most $\frac{1}{B_1}$th for large constant B_1) of the total fractional demand (i.e., $\sum_{v \in G_1} f_v$) on nodes in G_1.*

Since G is bounded degree, the total profit from flow on edges between G_1, G_2 is thus at most the profit from flow paths entirely inside G_1. Hence, we can throw this away and charge it to the demands we will route inside G_1 in the one-time routing. The overall algorithm then recurses on G_2.

This first property is assumed but we only need it to bound the loss from throwing away flow that crosses the cut (i.e., goes between inside and outside the contour). The next property, however, is more intrinsic to developing our one-time routing algorithm. At this point, we assume that the 1-Cut Procedure has been applied, and hence we are working in the *hard case* where G_1 is a nontrivial 2-node-connected graph. In this case the existence of the contour implies that G_1 has an outside face which is a simple cycle C. The next property follows because the contour from [5] is what they call "good".

Property 2 *Let f be a multi-flow in the graph G_1 for some of the demands $s_i t_i$.*

G_1 is said to have the face-flow property FF *with constant B_2 if the terminals in G_1 can simultaneously route a $\frac{1}{B_2}$-fraction of their flow to C. That is, there is a multi-flow g from terminals to C where v routes f_v/B_2 amount to C. Moreover, each node $v \in C$ terminates at most one unit of g's flow and $g(e) \leq 1$ for each $e \in E(G_1)$.*

It follows that the claimed congestion 2 algorithm exists if we may prove the following.

[7] Informally, a contour is a closed simple curve in the plane; we refer the reader to texts in algebraic topology for more formality as needed.

Theorem 4.1 (One-Time Routing Algorithm) *Let G_C be a 2-node-connected planar graph whose outside face has a cycle boundary C. Let X be the terminals associated with a collection of demands in G_C for which we have a multi-flow f. Let $\Delta = (\sum_{v \in X} f_v)/2$ denote the total flow for pairwise demands induced by f. If this instance has Property* FF*, then there is a (polytime) algorithm which finds a congestion 2 integral routing of $\Omega(\Delta)$ of these demands in G_C.*

We mention that the presentation of our algorithm and its analysis are self-contained except for (i) the preprocessing steps and the 1-Cut Procedure (discussed above), (ii) the sparse cutting routine which yields the previous two properties (Lemma 3.1 [5]), and (iii) the Okamura-Seymour results from Sect. 5.1 [5,21].

5 The congestion 3 algorithm

In this section we prove a weaker version of Theorem 4.1 where the routing has congestion 3.

We let M^* be the constant $\max\{B_1, B_2\}$ and $\Delta = \frac{1}{2}\sum_{v \in G_C} f_v$. We compute a congestion 3 integer routing of $\Omega(\Delta)$ demands in G_C. We repeatedly throw away constant fractions of our demand. If at some point the demand becomes at most M (for some large $M \geq M^*$ which could be chosen later), then it is sufficient to simply route one demand. The difficult case is when this does not occur—so one could assume $\Delta \geq M$.

Our edge-disjoint routing in G_C is created in two phases. In the end, we wish to route many demand pairs $s_i t_i$ but in Phase 1 we only identify some "promising" terminals. These terminals are able to simultaneously route unit flows to distinct nodes on the outside face C. If terminal v routes its flow to a node $z \in C$, then z is called its *representative*. In Phase 2, we show that the representatives can be connected by disjoint paths via an "Okamura-Seymour" routing. The final routing is obtained by gluing together the paths found in the 2 phases.

5.1 Okamura–Seymour

An *Okamura–Seymour (OS) instance* is a pair of weighted graphs: (i) $F = (V(F), E(F))$ is a planar "supply" graph with outside face C and edge capacities $u(e) \geq 0$, and (ii) $H = (V(C), E(H))$ is a demand graph with demands $d(h) \geq 0$ for each $h \in E(H)$. The associated multi-flow problem is *feasible* if there is an assignment $f : \mathcal{P} \to \mathbb{R}_{\geq 0}$ with the properties (a) $\sum_{P \ni e} f(P) \leq u(e)$ and $\sum_{P \in \mathcal{P}_h} f(P) = d(h)$. Here $\mathcal{P}$ denotes the set of all simple paths in F, and $\mathcal{P}_h$ is the set of all simple paths joining the endpoints of h. The standard necessary condition for feasibility is the *cut condition*:

$$\sum_{h \in \delta_H(S)} d(h) \leq \sum_{e \in \delta_F(S)} u(e) \quad \text{for each proper subset } S \subseteq V(F). \tag{2}$$

We rely heavily on a classical theorem.

Theorem 5.1 (Okamura-Seymour [21]) *Let F, u, H, d be an OS instance. Then there is a feasible multi-flow for the instance if and only if the cut condition holds. Moreover, if u, d are integer vectors and $F + H$ is an Eulerian Graph, then the cut condition implies the existence of an integral multi-flow.*

We face the following interesting snag. We can find an OS instance G_C, H which satisfies the cut condition and the total demand $\sum_{h \in E(H)} d(h) = \Omega(\Delta)$. The issue is that we need a large integral routing but the demands are highly fractional. The Okamura-Seymour Theorem says nothing about this case. We use the following result from [5] which follows from their Theorem 3.5 (Abstract Capacity Ring Lemma).

Theorem 5.2 (All-or-Nothing Okamura-Seymour) *Let F, H, d be an OS instance which satisfies the cut condition for unit capacities. In addition F is 2-node-connected. Then there is a subgraph $H' \subseteq H$ which is a matching and such that $|E'| = \Omega(\sum_h d(h))$. Moreover, the unit-demand instance on H' satisfies the cut condition with F.*

5.2 Phase 1: clustering

We let g denote the multi-flow which routes f_v/M^* from each terminal to the face C (guaranteed by Property 2). The PLANE-EDP4 algorithm is based on a rooted clustering of all demands congestion 2. We adapt this to be an edge-disjoint clustering of *some* of the demands using Theorem 2.3.

Proposition 5.3 *There is an edge-disjoint rooted clustering $R_1, R_2, \ldots R_h$ which captures $\Omega(\Delta)$ of the pairwise demand in G_C. Moreover, each cluster contains a unique node of C.*

Proof In order to invoke Theorem 2.3 we consider a rooted clustering instance where the root can be viewed as a single node t which is adjacent to each node of C by a unit-capacity edge. Each demand pair s_i, t_i contributes two terminals and their associated demand value $d_i = f_v/M^*$. The flow g thus almost implies that this instance is edge-normalized. Indeed the flow on each edge is at most 1 except for the edges into t. This is because each node of C is the final destination of at most one unit of flow, but it may also be used as an internal node for other flows. We now trim away another constant fraction of the flow to deal with this.

We consider each demand ab one by one. Consider the flow paths of g (with total size f_a/M^*) from a, b to C. If any of these flow paths uses more than one node of C, we truncate it at the first such node v it visits. Thus we think of this flow now entering the root (i.e., outside face C) at the node v instead of its original destination node on C. 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 simply drop the remaining flow from ab that we are trying to re-route to v. We charge any such lost flow for ab to the previous demands which terminated v. Since each node v has degree at most 4, the total loss (and hence 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.

We may now directly apply the Fractional Clustering Paired Capture Theorem 2.3 to the resulting reduced flow vector g' (i.e., edge-normalized instance). By construction, each resulting cluster contains a unique node of C. □

5.3 Phase 2 and the overall routing

Let $R_1, R_2, \ldots, R_h$ be the edge-disjoint rooted clustering from Proposition 5.3. For simplicity, g still denotes the corresponding ("single-sink") multi-flow from terminals to the outside face C. By construction we have that for each demand $s_i t_i$ the total flow from s_i and t_i are the same. Let us denote this quantity by g_i and note that we may assume that $\sum_i g_i$ is at least a constant fraction of Δ (or else, we are in the fall out position of routing one demand). The reduction of flows in g must also be reflected by reducing flow values in f; so we assume that $f_{s_i} = f_{t_i} = g_i$.

It could be the case that lots of clusters already contain sibling pairs s_i, t_i. If this happens for enough of them, we can immediately route one demand in each such cluster. So we can assume that at least half the flow from f is between pairs which lie in distinct clusters. At another factor-2 loss we assume these are the only demand pairs with flow and we restrict attention to only the associated terminals.

For each terminal v, let $m(v)$ denote the unique node of C which is contained in v's cluster; call this v's *representative*. We create a demand edge h between $m(v)$ and $m(\sigma(v))$ of weight $d(h) = f_v/2$. The following is implicit in [5].

Proposition 5.4 *The OS instance $G_C, H/d$ satisfies the cut condition.*

Proof Let R denote the terminal pairs $m(s_i)m(t_i)$ which are separated by some cut $\delta(S)$. Note first that we can route $\hat{f} = g/2 + f/2$ in G_C. We also have that at least one of the pairs $m(s_i)s_i, s_i t_i, t_i m(t_i)$ is separated by the cut. In each case the flow $\hat{f}$ uses at least $d(s_i t_i)$ capacity in the cut. Hence $|\delta(S)| \geq \sum_{h \in R} d(h)$. □

We now apply Theorem 5.2 to $G_C, H/d$ to deduce the following.

Proposition 5.5 *There is a unit-demand graph $H' \subseteq H$ which satisfies the cut condition in G_C and $|E(H')| = \Omega(\sum_{h \in H} d(h))$. Moreover, H' is a matching.*

We call the edges of H' the *OS-selected* demands.

Phase 2 consists of routing the OS-selected demands. These satisfy the cut condition but to invoke the Okamura-Seymour Theorem we also need that $G_C + H'$ is Eulerian. To do this, let T be the odd-degree nodes in $G_C + H'$. Recall that for a graph F and even subset X of $V(F)$, an *X-join* is a subset of edges which induces a graph whose odd-degree nodes are the set X. We use the following well-known fact [11].

Fact 5.6 *If F is a connected graph and X is an even subset of $V(F)$, then $E(F)$ includes a X-join.*

This fact implies that there is a T-join in G_C (not just $G_C + H'$). This is because removing the edges of H' still leaves a connected graph G_C and since T is even (since the parity of odd-degree nodes in $G_C + H'$ is even). If J is such a T-join, then $G_C + H' + J$ is Eulerian. Hence we can now apply the Okamura-Seymour Theorem to obtain a routing for H' in $G_C + J$.

The overall routing is completed as follows. The path between some pair $ab \in H'$ uses 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 of OS-selected), and finish by following the path from $m(b)$ to b using b's cluster. This has congestion 3 since Phase 1 routing (paths a to $m(a)$ and b to $m(b)$) can be done with congestion 1 in G_C since we have edge-disjoint rooted clusters. For Phase 2, we have just shown that H' can be routed in $G_C + J$ (which needs at most double the capacity of G_C).

6 The congestion 2 algorithm

The congestion 3 algorithm has an OS-selected subset of demands H' which forms a matching on terminals Z which lie on the outside face cycle C of G_C. The algorithm routes the demands in H' using (i) one copy of G_C to perform Phase 1 routing of terminals to the face C, using the edge-disjoint clusters from Proposition 5.3, (ii) one copy of G_C to ensure that the Okamura-Seymour instance for H' satisfies the cut condition, and (iii) one copy of G_C to create a $T_{H'}$-join J so that $G_C + H' + J$ is Eulerian, i.e., where $T_{H'}$ is the set of odd-degree nodes in $G_C + H'$. We show that one may modify H' so that Phase 1 and the $T_{H'}$-join can share one copy of G_C. Hence we only incur congestion of 2.

For each $v \in Z$, let P_v be the path from v to $m(v)$ through v's cluster. We refer to these as *stub paths* and recall that Proposition 5.3 gives us $|P_v \cap V(C)| = 1$. These are the edges used in Phase 1 Routing. For any $X \subseteq H'$ define G^X to be the subgraph of G_C obtained by deleting the stub paths for demands in X. Note that if $G^{H'}$ were connected, then it would already contain a $T_{H'}$-join J by Fact 5.6. Hence as before we could route the OS-selected demands in $G_C + J$ by the Okamura-Seymour Theorem, since $G_C + H' + J$ is Eulerian. So let $C_1, C_2, \ldots C_r$ be the connected components of $G^{H'}$, $r > 1$. Since each stub path contains precisely one node of C, $V(C)$ is included in one of these components; we assume it is C_1 and we refer to this as the *C-component* or *outer face component*.

Call a component C_i *$T_{H'}$-even* (resp. *odd*) if $T_i = T_{H'} \cap V(C_i)$ has even (resp. odd) size. Suppose that each component is $T_{H'}$-even. Again by Fact 5.6 (taking $X = T_i$), each $G_i = G[C_i]$ contains a T_i-join. Hence the union of these joins gives a $T_{H'}$-join J and we are also done. Henceforth our objective is to fix the *$T_{H'}$-odd* components.

The strategy for doing this is to only route a subset H'' of the demands in H'. We denote by Z'' the terminals associated with a subgraph $H'' \subseteq H'$. We can use capacity in the stub paths $\{P_v : v \in Z \backslash Z''\}$ to build the desired $T_{H''}$-join. In other words, $G^{H''}$ is available since this is not needed for Phase 1 routing. Finding H'' requires *sacrificing* some demands $v\sigma(v)$. This means deleting the demand from H' and hence adding $P_v, P_{\sigma(v)}$ to $G^{H''}$. The more sacrificed demands, the larger the graph $G^{H''}$. We grow this graph until all its components are $T_{H''}$-even.

Since sacrificing a demand $v\sigma(v)$ means removing it from H', this can also change the odd-degree nodes in $G_C + H''$ compared to $G_C + H'$. In other words, $T_{H'}, T_{H''}$

may differ. The only differences, however, are for nodes in C. Hence for $i > 1$ the component C_i is $T_{H'}$-odd if and only if it is $T_{H''}$-odd. It follows that if there is some $T_{H''}$-odd component, then there is at least one such C_i with $i > 1$. This implies the following, where T denotes the odd-degree nodes in G.

Fact 6.1 *If $G^{H''}$ has no T-odd components, then $G^{H''}$ contains a $T_{H''}$-join.*

Before describing the algorithm to produce H'', we first perform some preprocessing of the instance G_C, H'. For a stub path P_v which does not have both ends in C_1, we let $C(v)$ denote the non-outer face component which contains its non-C endpoint. We call a component $C_i \neq C_1$ *small* if it contains only one terminal v. We call such a component *isolationist* if the path P_v "cuts off" the component, i.e., $\delta_{G_C}(C_i) \subseteq E(P_v)$. We show that we can modify our instace so no such components exist.

Lemma 6.2 *For a 2-connected instance G_C and OS-selected demands H', we may modify the stub paths so that there are no isolationist components.*

There is one other configuration we can rule out before constructing H''. We call a demand $v\sigma(v)$ *cozy* if $C(v)$ (say) is small and the only other stub path to intersect $C(v)$ is $P_{\sigma(v)}$. Clearly we can route all cozy demands on edge-disjoint paths in G_C. Hence we can assume that the number of cozy demands is small, i.e., not a constant fraction of $|H'|$. Henceforth we assume that no such demands exist.

We now prove the following theorem which immediately implies Theorem 1.1.

Theorem 6.3 *There is an algorithm which takes as input a 2-connected instance G_C and OS-selected H' without cozy demands and such that $G^{H'}$ has no isolationist components, and computes $H'' \subseteq H'$ such that $|E(H'')| = \Omega(|E(H'|)$ and $G^{H''} = G_C \backslash (\cup_{v \in Z''} P_v)$ includes a $T_{H''}$-join.*

Proof The algorithm has two phases. In the first it starts with $H'' = H'$ and repeatedly looks for demands $v\sigma(v)$ to *sacrifice* in order to merge certain T-odd components with the outside face component. Sacrificing a demand means to remove it from H'' and hence add P_v (and also $P_{\sigma(v)}$ since it can only help) to $G^{H''}$. We then enter a second *protection phase* where we select some of the demands remaining in H'' to keep permanently. The algorithms stops when every demand has either been sacrificed or protected (defined below).

Let us first analyze the structure of a T-odd component C_i, $i > 1$ in $G^{H''}$. By definition of T and since $C_i \cap C = \emptyset$, we have $|\delta_{G_C + H'}(C_i)| = |\delta_{G_C}(C_i)|$ is odd; from now on we are working in G_C and drop the subscript δ_{G_C}. Since any path P_v with $v \notin C_i$ contributes an even number of edges to $\delta(C_i)$ (it starts and ends outside C_i), there are an odd number of P_y's with $y \in C_i$ for any T-odd component. We call C_i *big* if this number is at least 3. Otherwise it is *small* and so there is a unique stub path starting in C_i.

We call P_v *long* if it intersects at least 10 non-outer face components which are non-empty, i.e., containing at least one terminal. Note that P_v could have both endpoints in C_1 (i.e., $C(v) = C_1$) but may still intersect other components. If there is a long stub path, then we sacrifice the demand $v\sigma(v)$, and thus add P_v and $P_{\sigma(v)}$ to $G^{H''}$. This merges all of the components on P_v with the outer face component C_1. We repeat this until there are no long stub paths. Note that if we sacrificed k stub paths in this way, then we have merged at least $10k$ components with C_1. Thus we added at least $10k$ new terminals into this component. Therefore $10k \leq 2|H'|$ and so $k \leq \frac{|H'|}{5}$. It follows that the reduced demand graph H'' still has $\Omega(|H'|)$ demands which are not sacrificed.

We now enter the protection phase where the algorithm greedily considers the remaining demands $ab \in H'' \subseteq H'$ for permanent retention. To be retained, we must protect it from being sacrificed when we process demands after it.

We consider the components $C(a), C(b)$ containing a, b one at a time (it may be that $C(a) = C(b)$, that does not affect the argument.) If $C(a)$ is T-even or $C(a) = C_1$, we do nothing. Otherwise, $C(a)$ is T-odd and is disjoint from $V(C)$. In this case we try to merge $C(a)$ with C_1. If $C(a)$ is big, then we try to pick some path $P_z \notin \{P_a, P_b\}$ which also starts in $C(a)$. If there is such a P_z which is not protected, then we *sacrifice* $z\sigma(z)$ and remove it from H'', hence adding $P_z, P_{\sigma(z)}$ to $G^{H''}$. This merges $C(a)$ with the component C_1 in $G^{H''}$. If this is not possible, then some stub path which started in $C(a)$ must have already been protected, and hence $C(a)$ must already have merged with C_1, a contradiction.

Now consider the case where $C(a)$ is small. Since $C(a)$ is not isolationist, there is some P_z with $z \notin C(a)$ such that P_z passes through $C(a)$. Moreover, since we assumed there are no cozy demands in H', there is at least one such z with $z \neq b$. If $z\sigma(z)$ was not protected, then we sacrifice it and thus $C(a)$ is merged. Otherwise, we must sacrifice ab itself and we keep track of this by *charging* $z\sigma(z)$. We repeat the process for $C(b)$ and if we are successful, then each of $C(a), C(b)$ is either merged or T-even. The demand ab is then said to be *protected*. Note that when a demand ab is protected, it sacrifices at most 2 other demands to achieve this.

After completing the above process, H'' consists only of protected demands. Each such demand sacrificed at most 2 others to become protected, but it also may have been charged by demands processed after it which failed to be protected. If a demand ab is charged, then it is due to some terminal v whose component $C(v)$ (small in fact) intersected either P_a or P_b. Since P_a, P_b were not long, ab could be charged by at most 18 such demands. It follows that the number of sacrificed demands in the protection phase is at most $20|H''_{init}|$ where H''_{init} is the demand graph at the beginning of this phase. Hence for the final H'', $|H''| = \Omega(|H'|)$.

We now claim that every component in $G^{H''}$ is $T_{H''}$-even. By Fact 6.1, it is sufficient to show there is no T-odd component. If there is such a component, then there are at least two and hence C_i is T-odd for some $i > 1$. Therefore, there must be a path P_v with $v \in C_i$ which was not sacrificed and $C(v)$ was not merged. However, the algorithm should have processed $v\sigma(v)$ as a candidate to be protected, a contradiction. □

6.1 Preprocessing to eliminate isolationist components

In this section, we establish Lemma 6.2. Let $C_1, C_2, \ldots, C_k$ denote the components of $G^{H'}$ obtained by deleting the edges in all stub paths. We assume C_1 is the component which contains the outside face. A component C_i is *isolationist* if it is T-odd and

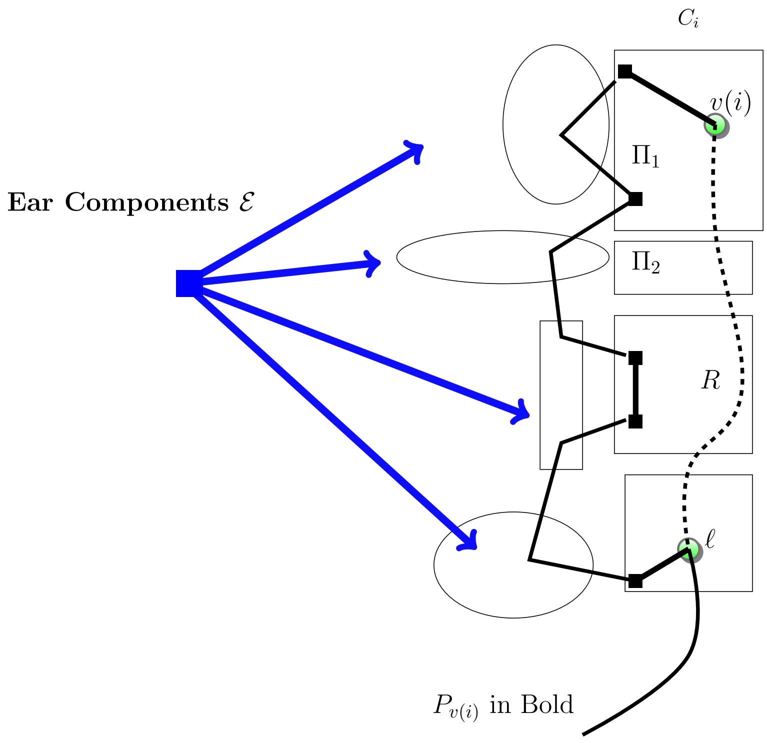

Fig. 4 Ears $\mathcal{E}$ and replacement path P^*

contains a unique terminal v and $\delta(C_i) \subseteq E(P_v)$. We now describe a preprocessing procedure which re-arranges some of the stub paths to eliminate such components.

Consider an isolationist component C_i, and the path $P_{v(i)}$ starting in C_i. Since G_C is 2-connected, and $\delta(C_i) \subseteq E(P_{v(i)})$, we have $|E(P_{v(i)}) \cap \delta(C_i)| \geq 3$. In traversing $P_{v(i)}$ from $v(i)$ to C, let ℓ be the last node contained in C_i. By definition $G_C[C_i] \backslash E(P_{v(i)})$ is connected and so there is a subpath R in this graph which joins $v(i)$ and ℓ. In particular, R is edge-disjoint from $P_{v(i)}$. We define a *replacement* path P^* to be the path which follows R to ℓ and then follows $P_{v(i)}$. The intent is to use this as $v(i)$'s path to the outside face.

We call an *ear* of $P_{v(i)}$ (w.r.t. C_i) a subpath Q, of length at least 2, whose starting and endpoints u, v say, lie in C_i, but whose internal nodes are outside C_i. A *Q-ear component* is some component C_j (disjoint from C_i) which contains an internal node of Q. We let $\mathcal{E}$ denote the set of Q-ear components for some ear Q of $P_{v(i)}$.

We now consider the effect on $G^{H'}$ by using P^* instead of $P_{v(i)}$, i.e., by removing $E(R)$ from $G^{H'}$ and then adding the edges of $P_{v(i)}$ on the subpath, call it F, from $v(i)$ to ℓ. Note first, that removal of R may break C_i into connected components, say $\Pi_1, \Pi_2, \ldots, \Pi_t$ (they may re-merge when F is added back in). Without loss of generality, $v(i)$ lies in Π_1. By definition of isolationist, Π_1 is the only T-odd component amongst these Π_k's. Namely, there is no stub path starting in any other Π_k and hence each of these Π_k's induces an even cut (recall T are the even degree nodes of G_C which do not lie on C). The notion of isolationist also implies that each $\Pi_k, k > 1$ the edges of $\delta_{G_C}(\Pi_K) \subseteq E(R) \cup E(P_{v(i)})$ (no other stub path can enter Π_k). Finally, after adding F back into $G^{H'}$, the new component C'_i containing $v(i)$ is obtained as follows. The nodes of Π_1 are merged with any components of $\mathcal{E}$ together with any even components Π_j touched by F—see Fig. 4. If this new component becomes T-even or big, then we have reduced the number isolationist components by 1. So suppose this new component is still isolationist.

Note that the only other components modified or created are even components induced by Π_k's which did not touch F. Hence if C'_i is no longer isolationist, we may just repeat this process on the next isolationist component. So suppose that C'_i remains isolationist (and hence small). By definition, no stub path P_z for $z \neq v(i)$ could have entered any of the ear components. But then it follows that $|\delta_{G_C}(C_i \cup C'_i)| = 1$, contradicting 2-node connectivity. □

7 Conclusions

A very interesting question is to extend the methods to get a constant factor approximation for the problem in planar graphs when the commodities have associated weights. It seems probable that this is related to a well-known question on the existence of Bi-Lipschitz embeddings of planar metrics into ℓ_1. It is also possible that the current techniques can be extended to give an $O(1)$-approximation for Eulerian, planar instances. This is slightly less natural from the approximation perspective, but would give a strictly stronger result.

Since the preliminary version of this paper [23] it was conjectured that the natural LP has a constant integrality gap with congestion $O(1)$ (or even 2) for any minor-closed family of graphs [6]. This remains open but applying the results from this paper they have shown that bounded genus graphs have a constant integrality gap with congestion 3.

Acknowledgements The authors are grateful for excellent feedback from three anonymous referees. The second author thanks Chandra Chekuri for many stimulating conversations on this and related topics. This work has been primarily supported by an NSERC Discovery Grant.

References

1. Andrews, M.: Approximation algorithms for the edge-disjoint paths problem via Räcke decompositions. In: 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, pp. 277–286 (2010)
2. Andrews, M., Chuzhoy, J., Guruswami, V., Khanna, S., Talwar, K., Zhang, L.: Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. Combinatorica **30**(5), 485–520 (2010)
3. Chekuri, C., Khanna, S., Shepherd, F.B.: Edge-disjoint paths in planar graphs. In: 45th Annual IEEE Symposium on Foundations of Computer Science, 2004. Proceedings, pp. 71–80 (2004)

4. Chekuri, C., Chuzhoy, J.: Maximum node-disjoint paths with congestion 2. Personal communication (2016)
5. Chekuri, C., Khanna, S., Bruce Shepherd, F.: Edge-disjoint paths in planar graphs with constant congestion. In: STOC '06: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, pp. 757–766. ACM, New York, NY (2006)
6. Chekuri, C., Naves, G., Bruce Shepherd, F.: Maximum edge-disjoint paths in k-sums of graphs. In: International Colloquium on Automata, Languages, and Programming, pp. 328–339. Springer (2013)
7. Chen, J., Kleinberg, R.D., Lovász, L., Rajaraman, R., Sundaram, R., Vetta, A.: (Almost) tight bounds and existence theorems for single-commodity confluent flows. J. ACM **54**(4), 16 (2007)
8. Chuzhoy, J.: Routing in undirected graphs with constant congestion. In: Proceedings of the 44th Symposium on Theory of Computing, pp. 855–874. ACM (2012)
9. Chuzhoy, J., Li, S.: A polylogarithimic approximation algorithm for edge-disjoint paths with congestion 2. arXiv preprint arXiv:1208.1272 (2012)
10. Chuzhoy, J., Kim, D.H.K., Nimavat, R.: New hardness results for routing on disjoint paths. arXiv preprint arXiv:1611.05429 (2016)
11. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. Wiley-Interscience, Hoboken (1997)
12. Dinitz, Y., Garg, N., Goemans, M.X.: On the single-source unsplittable flow problem. Combinatorica **19**(1), 17, 01–41 (1999)
13. Donovan, P., Shepherd, B., Vetta, A., Wilfong, G.: Degree-constrained network flows. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of computing, pp. 681–688. ACM (2007)
14. Frank, A.: Packing paths, cuts, and circuits—a survey. In: Korte, B., Lovász, L., Prömel, H.J., Schrijver, A. (eds.) Paths, Flows and VLSI-Layout, pp. 49–100. Springer, Berlin (1990)
15. Frank, A.: Connections in Combinatorial Optimization. Oxford University Press, Oxford (2011)
16. Garg, N., Vazirani, V.V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica **18**(1), 3–20 (1997)
17. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., Yannakakis, M.: Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. J. Comput. Syst. Sci. **67**(3), 473–496 (2003)
18. Kawarabayashi, K., Kobayashi, Y.: All-or-nothing multicommodity flow problem with bounded fractionality in planar graphs. SIAM J. Comput. **47**(4), 1483–1504 (2018)
19. Kleinberg, J., Tardos, É.: Approximations for the disjoint paths problem in high-diameter planar networks. In: Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, pp. 26–35. ACM (1995)
20. Kleinberg, J.M.: 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, pp. 627–636 (2005)
21. Okamura, H., Seymour, P.D.: Multicommodity flows in planar graphs. J. Combin. Theory Ser. B **31**(1), 75–81 (1981)
22. Raghavan, P., Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica **7**, 365–374 (1987)
23. Seguin-Charbonneau, L., Shepherd, F.B.: Maximum edge-disjoint paths in planar graphs with congestion 2. In: 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 200–209. IEEE (2011)