Edge-Disjoint Paths in Planar Graphs with Constant Congestion*

Chandra Chekuri† Sanjeev Khanna‡ F. Bruce Shepherd§

November 8, 2006

Abstract

We study the maximum edge-disjoint paths problem in undirected planar graphs: given a graph G and node pairs (demands) s_1t_1, s_2t_2, ..., s_kt_k, the goal is to maximize the number of demands that can be connected (routed) by edge-disjoint paths. The natural multicommodity flow relaxation has an $\Omega(\sqrt{n})$ integrality gap where n is the number of nodes in G. Motivated by this, we consider solutions with small constant *congestion* $c > 1$; that is, solutions in which up to c paths are allowed to use an edge (alternatively, each edge has a capacity of c). In previous work we obtained an $O(\log n)$ approximation with congestion 2 via the flow relaxation. This was based on a method of decomposing into *well-linked* subproblems.

In this paper we obtain an $O(1)$ approximation with congestion 4. To obtain this improvement we develop an alternative decomposition that is specific to planar graphs. The decomposition produces instances that we call *Okamura-Seymour* (OS) instances. These have the property that all terminals lie on a single face. Another ingredient we develop is a constant factor approximation for the all-or-nothing flow problem on OS instances via the flow relaxation.

1 Introduction

1.1 Edge-Disjoint Paths, Cut Condition, and Multicommodity Flow

In this paper we study the *edge-disjoint path* problem (EDP) in undirected graphs. We are given a supply graph $G = (V, E)$ and a demand graph $H = (V, F)$. We sometimes write the demand set F as a set of node pairs $\mathcal{T} = \{s_1t_1, s_2t_2, \ldots, s_kt_k\}$. We sometimes refer to the s_i, t_i's as *terminals*. The objective is to determine if we can connect the pairs in F via edge-disjoint paths. If such a collection exists, the demands are said to be *routable*. In the maximum edge-disjoint path problem (MEDP), the objective is to find a maximum routable subset of F (or $\mathcal{T}$). More generally, we sometimes allow each demand to have a specified demand amount d_i, and supply edges to have capacities; we also may relax the notion of routability to suit our purposes. For instance, we may allow the demand to be fractionally routed (*multicommodity flow*) or we may require that the whole demand be routed on a single path (*unsplittable flow*).

The problem of determining whether a set of demands is routable is one of the classical NP-hard problems in combinatorial optimization [10, 15]. There are two natural necessary conditions that are employed in tackling this feasibility problem. The first is the *cut condition*: for every proper subset $S \subseteq V$, $|\delta_G(S)| \geq |\delta_H(S)|$. If demand edges have values, then the right-hand side becomes $\sum_{f \in \delta_H(S)} d_f$,

*A preliminary version of this paper appeared in *Proc. of ACM STOC*, 2006.

†Department of Computer Science, University of Illinois, Urbana, IL 61801. Email: chekuri@cs.uiuc.edu.

‡Department of Computer and Information Science, University of Pennsylvania, Philadelphia PA. Email: sanjeev@cis.upenn.edu.

§Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974. Email: bshep@research.bell-labs.com.

and if edges have capacities, $\delta_G(S)$ is altered accordingly. Here we use $\delta_G(S)$ to denote the set of edges in a graph G with exactly one endpoint in S. The second necessary condition is the existence of a *multicommodity flow* (MCF) for the demand graph H. That is, an assignment of nonnegative flow $f(P)$ to paths P that obey the following capacity and demand constraints: (1) $\sum_{P:e\in P} f(P) \leq c(e)$ and (2) $\sum_{P\in\mathcal{P}_i} f(P) = 1$. We let $\mathcal{P}_i$ denote the paths joining s_i and t_i in G and let $\mathcal{P} = \cup_i \mathcal{P}_i$. In (1) $c(e)$ denotes the capacity of edge e.

Obviously, if F is routable, then it has a multicommodity flow, and the existence of a flow implies that the cut condition holds. However, there are instances where the cut condition holds but for which there is no flow, and instances where a flow exists but for which F is not routable. Not surprisingly, research has tended to bifurcate into two categories. One is to examine special cases where the cut condition (or more general conditions such as the existence of a flow) guarantees solvability of the disjoint path problem. A second direction is to quantify the degree to which routability (fractional or integral) diverges from the cut condition.

In the first category, there are two archetypal examples that play a central role in the current study. One is the max-flow-min-cut theorem for single-source multicommodity flow. Here we have a source node s and terminals $t_1, t_2, \ldots, t_k$ (possibly also with associated integer demands d_i). Results of [11, 24] then show that the cut condition is sufficient for the simultaneous routability of the demands. The second is the Okamura-Seymour Theorem which states that the cut condition is sufficient for solvability of multicommodity flow for any instance where G is planar and all demand edges have both ends on a single face of G, in particular on the outer face.

In the second category, the notion of *concurrency* plays a rôle in measuring so-called *flow-cut gaps*. Given an instance of EDP or MCF that satisfies the cut condition, one asks for the largest value $\lambda > 0$ such that there exists a multicommodity flow that simultaneously routes λd_f for each demand edge f. The inverse of the worst case value of λ over all all instances in G is the flow-cut gap for G. In [22], a first positive breakthrough was given in bounding the flow-cut gap for general undirected graphs G; a bound of $O(\log n)$ was established for uniform instances of MCF. For the non-uniform case a similar bound was shown in [3, 23]. In planar graphs, for uniform multicommodity flow, the flow-cut gap was shown to be $O(1)$ [16], while for non-uniform multicommodity flow the current best upper bound on the gap is $O(\sqrt{\log n})$ [27].

1.2 Maximum Edge-Disjoint Paths Problem

In tackling the throughput maximization problem MEDP, one has a natural LP relaxation. Assuming the unweighted version, we seek to maximize $\sum_P f(P)$ subject to the constraints (1) and (2) above, where we relax the latter to inequality. If OPT is the optimal value of the LP, then we seek to find a feasible integral solution that guarantees a large fraction of OPT.

Unfortunately, the integrality gap for this LP is $\Omega(\sqrt{n})$, even in the case of planar graphs [14]. An $O(\sqrt{n})$ upper bound on the integrality gap, even in general graphs, is shown in [8]. In contrast, known inapproximability bounds are exponentially smaller: only recently a first super-constant lower bound was given in [1]: unless $NP \subseteq ZPTIME(n^{\text{polylog}(n)})$ there is no $O(\log^{\frac{1}{3}-\epsilon} n)$ approximation for MEDP (this hardness factor is improved to $O(\log^{\frac{1}{2}-\epsilon} n)$ in [2]).

The difficulty of MEDP stems from two seemingly distinct aspects: one arising from the gap between fractional and integral flows, and the other stems from the selection of which subset of demands to try to route. When G itself is a tree, the first complication disappears and one may focus on the issues involved in choosing a most profitable routable subset of demands. Even on trees this problem is APX-hard [14]. A 2 approximation for the cardinality case was shown in [14] and a 4 approximation for the weighted case is shown in [4]. To understand the difficulty of the subset selection problem, a relaxation of MEDP was proposed in [4]: to find a maximum fractionally routable

subset of the demands. This *all-or-nothing* multicommodity flow problem (ANF) was studied in [5]. The natural LP relaxation for ANF is the same as that for MEDP. In [5], it was shown that this LP has a poly-logarithmic integrality gap. We mention that the inapproximability bounds of [1, 2] also apply to ANF, and hence the upper and lower bounds are separated by only a poly-logarithmic factor.

The focus of the present paper is a second relaxation of MEDP, that of allowing *congestion* on the edges of G. Here, we seek a sizable fraction of OPT by allowing the routable set to use each edge up to some constant number of times (alternatively, each edge is endowed with some constant capacity). Even with congestion c, it is shown in [2] that unless $NP \subseteq ZPTIME(n^{\text{polylog}(n)})$ no $O(\log^{\frac{1-\epsilon}{c+1}} n)$ approximation is possible. Similar lower bounds are also shown for the integrality gap of the flow relaxation [2]. However no super-constant integrality gap is known for planar graphs if congestion 2 is allowed. In contrast the best upper bound known was $O(\sqrt{n})$ until recent work [6, 7] which obtained a bound of $O(\log n)$ with congestion 2. Our main result, stated below, considerably strengthens this result.

Theorem 1.1 *For the MEDP problem in a planar graph, there is a polynomial time algorithm to route* $\Omega(\text{OPT})$ *demand pairs with congestion* 4*, where* OPT *is the value of the multicommodity flow relaxation.*

We have given only the briefest overview to set the context and motivation for proving the above result. Some other related work in the area is discussed in the next subsection.

1.3 Outline

We outline the main ingredients of the proof Theorem 1.1, and how it differs from the framework used in [5, 6, 7] for solving the two throughput maximization problems ANF and MEDP. In the following we assume, without loss of generality, that the edge set F of the demand graph $H = (V, F)$ induces a matching M. We also let X denote the endpoints of M, the *terminals* of the instance.

Decompositions into Well-Linked Sets: Our previous approach to producing a large routable set was based on finding "well-connected" sets of terminals. A set X of terminals is *well-linked* if for each $S \subseteq V$ with $|S \cap X| \leq |(V \setminus S) \cap X|$, we have $|\delta(S)| \geq |S \cap X|$. The relevance of well-linked sets for edge-disjoint path problems in planar graphs is that a well-linked set implies the existence of a grid minor of size $\Omega(|X|)$ [29]. Hence, if edge congestion 2 is allowed, one may satisfy a constant fraction of any matching on the terminals, by routing through the grid minor [6].

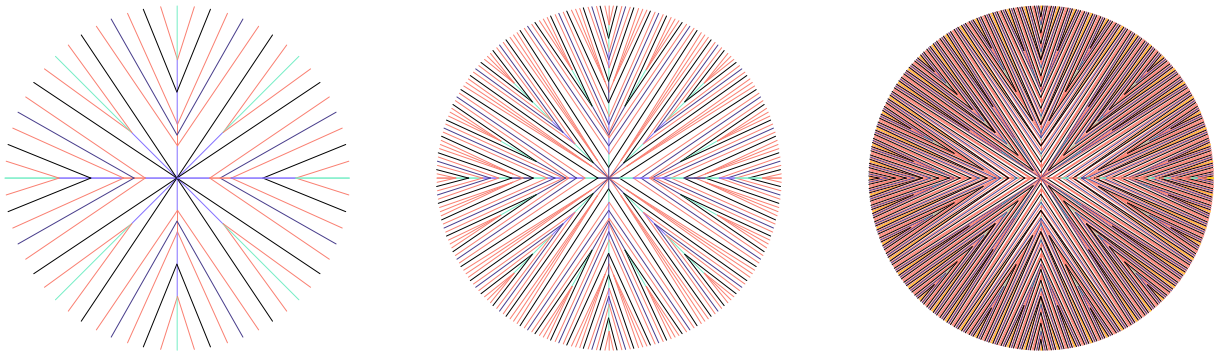

Figure 1: The duals D_6, D_8, D_{10} of the 6,8, and 10-dimensional recursively divisible planar graphs.

We consider one approach to find a well-linked subgraph, starting from a fractional flow. We nominally treat terminals as having "weight" one, but in general this weight represents the total

flow they originate in the fractional solution. We find an induced subgraph $G[S]$ that contains say p terminals from the demand pairs while $|\delta_G(S)|$ is small compared to p, say ϵp. We say S has a short boundary with respect to the terminals inside. Note that if the terminal set is not well-linked in the original graph, then there is a sparse cut S with respect to the terminals, and this identifies a short-boundary set. If we choose S carefully and the terminals in $G[S]$ turn out to be (approximately) well-linked, then the total flow lost by removing edges of $\delta(S)$, can be *charged* to $G[S]$ and we can recurse on $G[V \setminus S]$. This scheme can be applied to obtain a well-linked decomposition with only a constant factor loss if G happens to be a capacitated tree: choose S to be a *minimal* set with a short boundary. (This observation combined with Räcke's [26] hierarchical decomposition of graphs can be used to obtain a well-linked decomposition with a poly-logarithmic approximation ratio. This is implicitly shown in [5].) In general graphs, or even planar graphs, choosing a minimal short boundary set S does not guarantee well-linkedness of the terminals in $G[S]$. In [7] a simple recursive scheme was used. For some ϵ that is sufficiently small (inverse poly-logarithmic), we find a set S that is ϵ-short and recurse on $G[S]$ and $G[V \setminus S]$.

It is natural to ask whether the above recursive procedure yields a well-linked decomposition that retains a constant factor of the flow if ϵ is some fixed constant. In particular can this be done for planar graphs? If this is the case, then we would obtain a constant factor approximation for MEDP in planar graphs with congestion 2. Unfortunately there are instances where using the recursive procedure (for any constant ϵ) will not retain even a logarithmic fraction of the initial flow. Such instances have a property we call *recursive divisibility* with respect to a set of terminals X. More concretely, there is a binary tree whose leaves are the nodes of V and for each internal node u, with subtrees T_l, T_r we have that the number of edges of G with one end in T_l and the other in T_r is at most $\epsilon \min\{|T_l \cap X|, |T_r \cap X|\}$. In other words, this tree represents a *sparse cutting procedure* where each cut produced along the way is "ϵ-sparse" relative to the original terminals. In Section 4 we show planar graphs that are recursively divisible for any constant $\epsilon > 0$. This shows that there are some "bad" well-linked cutting decompositions. We believe this suggests that well-linked decompositions might result in super-constant approximation in planar graphs – see Figure 1.3 for a picture of the duals of graphs in this class. In general graphs more formal negative results are possible. One can produce recursively divisible for $\epsilon = O(1/\log n)$ and moreover show that no well-linked decomposition produces better than a logarithmic approximation. The following theorem[1] gives a lower bound in general graphs. Precise definitions of a well-linked decomposition and the distinction between cut and flow well-linkedness can be found in [9, 7].

Theorem 1.2 ([9]) *There are instances of EDP on a graph with n nodes, with* OPT *denoting the value of the multicommodity flow relaxation such that any cut-well-linked (and hence any flow-well-linked) decomposition retains at most $O(\frac{\text{OPT}}{\log n})$ demand pairs from the matching.*

Outline of the New Algorithm: The description above shows that in order to produce a constant factor result for MEDP in planar graphs, we might need to diverge from the framework of finding well-linked subgraphs. As before, we continue to seek a decomposition into "well-connected" pieces. However, our notion of well-connected is weakened considerably. In a way, this is not surprising since well-linked sets admit good routings for *any* matching on the terminals inside the well-linked piece $G[S]$. In the new framework, the definition of well-connected is more tied to the specific set of demands for an instance.

The new algorithm looks for a minimal subset S such that $|\delta(S)|$ is small (at most some constant factor $\epsilon < 1$) relative to the total fractional flow in $G[S]$. We then show how to pick a subset of the

[1] A proof of this theorem appeared in the conference version of this paper [9]. This will be included in the journal version of [7].

demands with both ends in S that can be routed. This is achieved as follows. First, using minimality of S one can show that terminals in S can be fractionally routed to the outside face of $G[S]$; call this the *to-the-face routings*. Second, this routing is used to set up a feasible auxiliary multicommodity flow instance for demand pairs of nodes on the outside face of $G[S]$. The flow between distinct pairs are assorted values in the range $[0, 1]$. These are called OS instances (Okamura-Seymour). For the OS instance, we then solve an all-or-nothing flow problem, that is, for each pair its flow becomes either 0 or 1. We show that such a conversion is possible with only a constant factor loss in the total fractional flow. We then apply the Okamura-Seymour Theorem for the resulting demand pairs, to obtain a half-integral flow, thus corresponding to routing with congestion 2. Finally, this routing is stitched together with the to-the-face routing to satisfy a large fraction of our original demands. With some care, this can be achieved with an additional congestion of 2, to give the overall congestion bound of 4.

Other Related Work: Disjoint paths and routing problems have an extensive literature. We do not attempt to give a comprehensive overview and highlight here only the closely related work. We suggest [13, 30, 17, 21] for extensive introductions to the algorithmic work and some recent papers [6, 7, 18] for other pointers. The seminal work of Robertson and Seymour on graph minors, and their polynomial time algorithm for the disjoint paths problem for a fixed number of demand pairs [28], has had enormous impact on subsequent research. For planar graphs the first non-trivial algorithms for MEDP were given by Kleinberg and Tardos [19, 20] who considered grids and grid-like graphs. It led them to conjecture that poly-logarithmic approximation algorithms exist for all even-degree planar graphs. Implicitly the conjecture implied the existence of a poly-logarithmic approximation for planar graphs with congestion 2. This was achieved using well-linked decompositions introduced in [6, 7]. Kleinberg [18] obtained an $O(\log^2 n)$ approximation for all even-degree planar graphs also using such decompositions. In a variety of multicommodity flow problems Eulerian instances admit integral solutions where one might only suspect half-integral solutions – the reader is referred to [13, 30].

2 Preliminaries

Simplifying the Input Instance: The input consists of an undirected planar graph $G = (V, E)$ with integer edge capacities and a set of node pairs (called *demands*) $s_1t_1, s_2t_2, \ldots, s_kt_k$. We assume without loss of generality that each node in G has degree at most 4 and all edges have unit capacity. We refer the reader to [13, 6] for more details on how a given instance can be transformed to the bounded degree case while preserving planarity. The transformation increases the size of the graph to $\sum_e c(e)$. Note that this need not be strongly polynomial in the input size. However, as we indicate later, when using the LP relaxation, it is sufficient to assume $\sum_e c(e)$ is polynomial in n. We also assume that the node pairs are induced by a matching M on the terminal set X.

Multicommodity Flow LP Formulation: For the given instance with $\mathcal{T} = \{s_1t_1, s_2t_2, \ldots, s_kt_k\}$, we let $\mathcal{P}_i$ denote the paths joining s_i and t_i in G and let $\mathcal{P} = \cup_i \mathcal{P}_i$. The nodes s_i, t_i are called *terminals*. The following multicommodity flow relaxation is used to obtain an upper bound on the number of

demand pairs from $\mathcal{T}$ that can be routed in G.

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

For each path $P \in \mathcal{P}$ we have a variable $f(P)$ which is the amount of flow sent on P and x_i denotes the total flow sent on paths for pair i. We let $\bar{f}$ denote the flow vector with a component for each path P. We let $val(\bar{f})$ denote the value $\sum_i x_i$. We let OPT denote the value of an optimum solution to the relaxation.

Call a path P *fractionally routed* if $f(P) \in (0,1)$, otherwise $f(P) \in \{0,1\}$ and P is *integrally routed*. If the total flow routed on integrally routed paths is more than OPT/2, then we already obtain a 2-approximation. Thus the interesting and difficult case is when the fractionally routed paths contribute almost all the value of OPT. From standard polyhedral theory the number of fractionally routed paths in a basic solution to the LP above is at most m. Therefore we can assume that $c(e) \leq m$ for all $e \in E$. By making parallel copies of edges, in the following, we assume that G has only unit capacity edges. We can do the transformation to the bounded degree case (mentioned above) after solving the LP and this ensures that the resulting graph has size polynomial in n.

For a terminal $v \in X$, we let $f_v := x_i$ where $v \in \{s_i, t_i\}$ is the *marginal flow* at v. As the algorithm progresses, we are interested in fractional solutions restricted to some subgraph and hence some of the original flow is "lost". We implicitly work with the induced flow-path decomposition for such flows. Given a current flow, we use the weight function $b : X \to \mathcal{R}^+$ to denote the marginal flows (rather than renaming $\bar{f}$ every time we examine a new flow). Thus for a terminal $v \in X$, we let $b_v := x_i$ where $v \in \{s_i, t_i\}$ in the current flow; sometimes we also use $b_{s_i t_i} := x_i$. Thus for any such flow, b_v might be much smaller than its original marginal f_v. Throughout the algorithm we set up several single source flow instances to route flow from terminals to other nodes. In these instances we have an upper bound of b_v on the flow we allow from a terminal v. In such a context we refer to b_v as v's "flow".

Planar Embeddings and Contours: We assume that we have some fixed embedding of G on the sphere. We also fix some point ι of the sphere. A *G-curve* is a simple curve in the plane, whose image only intersects the embedding of G at nodes. If the image of its "beginning" and "endpoints" are the same, then we call such a *closed* simple curve a *G-contour*, or just a contour. The *length* of a G-curve C, denoted by $len(C)$, is the number of nodes of G whose embedding is contained in the image of the curve. (In the following, we abuse terminology and do not differentiate between nodes, edges, and their images in the embedding.) For any contour C, removing its image from the sphere produces two open regions (disks). We let $ins(C)$ (respectively $ext(C)$) denote the region not containing ι (respectively containing ι). Without loss of generality, there is a G-contour C whose length is 0 and whose interior contains the embedding of G. For a contour C we use the notation G_C to denote the subgraph of G induced by the nodes in the closed interior of C. See Figure 2.

3 The Approximation Algorithm

Given an instance of EDP on a planar graph G, we assume that the simplifications from the previous section have been performed. In particular, we are given some optimal multicommodity flow $\bar{f}$.

The heart of the approach is to find a contour C and its associated subgraph G_C with two properties. The first property is that the total flow from $\bar{f}$ contained inside G_C (that is, flow on paths that use only edges in G_C) is at least a constant fraction of flow from $\bar{f}$ that is either transiting through G_C or crossing G_C. The second property is that the terminals in G_C can *simultaneously* send a constant fraction of their flow in $\bar{f}$ to the boundary C. We ensure the first property in a simple way by choosing C such that $len(C)$ is a sufficiently small factor γ smaller than the total flow incident to terminals in G_C. Since the degree of a node on C is at most 4, if γ is chosen small enough, the total flow inside G_C is at least a constant factor larger than the flow crossing or transiting through G_C.

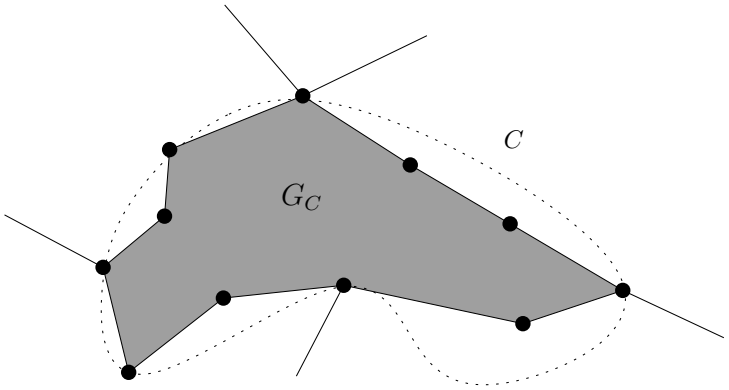

Figure 2: Contour

Suppose we find such a contour C. We restrict attention to the graph G_C and the flow from $\bar{f}$ that is induced in G_C. Using the second property, we set up a fractional instance of a disjoint path problem where all the terminals are on the outer face of G_C. We call such an instance an *Okamura-Seymour instance* (OS instance for brevity). For a demand pair st with both ends in G_C we set up a corresponding pair uv where $u, v \in C$. We show that for OS instances, the integrality gap of the LP (1) is a constant in the cardinality case. This is related to the pairing lemma of Frank [12] and strongly generalizes the result on trees [14]. The details are in Section 3.3. It is not sufficient to simply prove this. We also need to ensure that if a pair uv on C corresponding to a pair s_it_i is routed, then s_i and t_i can integrally route to u and v. To ensure this we set up the OS instance carefully, the details of which can be found in Section 3.2.

Thus we are able to integrally route (with constant congestion) a constant factor of the flow inside G_C. We then remove G_C from G and recurse on the remaining graph (and the induced flow). The first property of the contour C ensures that the total flow "lost" in removing G_C from G can be charged to the flow that is integrally routed in G_C. This process thus results in a constant factor approximation. Let $C_1, C_2, \ldots, C_\ell$ be the contours found in the process. The sub-graphs $G_{C_1}, G_{C_2}, \ldots, G_{C_\ell}$ are node-disjoint - in fact they are only edge-disjoint in the original graph before we do the transformation to the bounded degree case (see Section 2).

3.1 Finding and Routing to a Contour

In this section, we take as our starting point a fractional multicommodity flow vector $\bar{f}$ for the given instance such that for each commodity i, there is a total flow of x_i between s_i, t_i. For a terminal v let f_v denote the flow for the pair in which v participates. For any contour C, we denote by $T(C)$, the terminals that lie in G_C. We call a contour C *short* if its length (number of nodes) is at most $\lfloor \sum_{v \in T(C)} f_v/10 \rfloor$. Since the maximum degree of any node is at most 4, it follows that for any short

contour C, there is at least

$$\begin{aligned}\sum_{v\in T(C)} f_v/2 - 4len(C) &\geq \sum_{v\in T(C)} f_v/2 - 4\sum_{v\in T(C)} f_v/10 \\ &\geq \sum_{v\in T(C)} f_v/10\end{aligned}$$

flow contained entirely in G_C.

We call a short contour *good* if the terminals in $T(C)$ can each send $1/10$ of their flow to the contour. We make use of the following lemma to find a short good contour C.

Lemma 3.1 *Let G be a planar graph embedded in the plane and let $w : V \to [0,1]$ with $w(V) > 1$. Given $\alpha \in (1, w(V))$, there is a contour C computable in polynomial time with the following properties:*

- *C is α-tight, that is $len(C) = \lceil w(V(G_C))/\alpha \rceil$.*
- *C is α-good, that is, in the graph G_C, each node u can simultaneously send $w(u)/\alpha$ flow to C such that a node in C receives at most 1 unit of flow.*

Proof. We say that a contour C is *α-short* if $len(C) \leq \lceil w(V(G_C))/\alpha \rceil$. Given a contour C that is α-short but not α-tight, we can use simple greedy extension steps to find another contour C' that is contained in C and such that C' is α-tight. The two operations to achieve this are shown in Figure 3. In the first operation the length of the contour increases by one node. In the second operation we "shift" an edge which keeps the length of the contour the same but decreases the number of edges in the interior of C. These are performed as follows. If ever we can perform an operation of the second type we do so. If this is not possible, we try to perform the first operation. Here we shift the contour to touch a node whose image does not yet lie on C. If neither of these operations is possible, then it must be the case that every node of the graph G_C is contained in the image of C; this contradicts the fact that C is not α-tight. Thus after a polynomial number of steps we can find an α-tight contour starting from an α-short contour. Since we assumed that there is a 0-length contour whose interior contains G, we now know that there exists an α-tight contour.

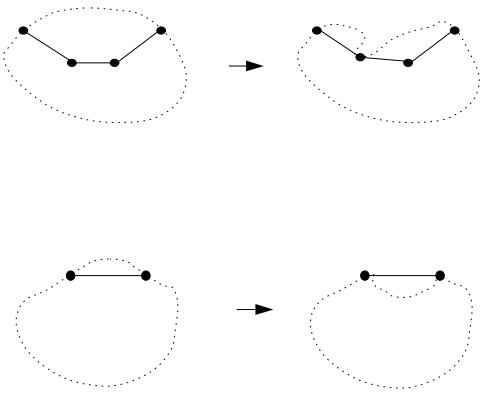

Figure 3: Shifting a contour.

Let C be an α-tight contour that minimizes the number of nodes in G_C. We claim that C is the desired contour. Suppose this is not the case. We consider a single-source flow problem $\mathcal{I}_C$ defined as follows. We create G' by adding to G_C, a new source node s and make it adjacent to each node v with

$w(v) > 0$ (terminal in the context of our routing problems) in the closed interior of C with edges of capacity $w(v)/\alpha$. We also add a node t and make it adjacent to each node v on C with a unit capacity edge. Let E' denote this set of additional edges. All edges in G_C are assigned unit capacity in G'. For a subset of edges F in G', let $\mathrm{cap}(F)$ denote the total capacity of edges in F in G'. Now the contour C is good if and only if the maximum s-t flow in G' equals $\sum_{u \in G_C} w(u)/\alpha$. If C is not good, then some s-t cut in G', say $\delta(S)$ (with $s \in S, t \notin S$), has capacity strictly less than $\sum_{u \in G_C} w(u)/\alpha$. Let $U_1, U_2, \ldots, U_c$ be the connected components of the induced graph $G'[S - s]$, and let U be the set of nodes of G_C not in any U_i. Note that $\mathrm{cap}(\delta_{G'}(S)) = w(U)/\alpha + \sum_i |\delta_{G'-s}(U_i)|$. Thus without loss of generality we have that $\mathrm{cap}(\delta_{G'-s}(U_1)) < w(U_1)/\alpha$. Since $G' - s$ is planar and U_1 induces a connected subgraph, there is a curve C' in the plane whose interior contains precisely the nodes of U_1 and only intersects edges from the set $\delta_{G'-s}(U_1)$. We may shift this curve to become a G-contour C'' which only intersects nodes of G_C and for which $V(G_{C''}) = U_1$. Since the edges of $\delta_{G'-s}(U_1)$ all had unit capacity in G', we also obtain that the length of C'' is less than $w(U_1)/\alpha$ nodes. Thus C'' is again α-short. This contradicts the minimality of C since U_1 must necessarily contain strictly fewer nodes from G_C. Otherwise, $U_1 = V(G_C)$, contradicting tightness of C.

The above proof can be made algorithmic as follows. The algorithm starts with a contour that contains the entire graph G in its interior. It applies the contour extension steps to find an α-tight contour C_0. It then checks if C_0 is a good contour by solving the single-source flow problem $\mathcal{I}_{C_0}$. If C_0 is good, then it is the desired contour. Otherwise $\mathcal{I}_{C_0}$ identifies a short contour C_1 that is strictly contained inside C_0. We then apply the above two steps, extension and solving the single-source flow problem to C_1, and so on. The number of nodes inside the contour decreases in each iteration and hence the algorithm terminates in a polynomial number of iterations. The process is guaranteed to succeed since a minimal α-tight contour is good, as shown above. □

We can use the above lemma with $w(v) = f_v$ if $v \in X$ and $w(v) = 0$ otherwise and set $\alpha = 10$. Let C be the contour obtained such that C is both α-tight and α-good contour. The contour C contains two types of terminals. A terminal whose mate is outside is termed *separated* and we eliminate these. Let Q be the remaining terminals; each terminal in Q has its mate also in Q (both are in G_C). Let R be the set of terminal pairs in G_C that are not separated. For a terminal $v \in Q$ let $b_v \leq f_v$ be the flow left in G_C. For a demand pair $st \in R$, let $b_{st} = b_s = b_t$ denote the flow between s and t that remains inside G_C. Let $p := \sum_{st \in R} b_{st} = \sum_{v \in Q} b_v/2$ be the total flow in G_C. As we saw earlier, $\sum_{v \in R} b_v/2 \geq \sum_{v \in T(C)} f_v/10$. Next we show that $\Omega(p)$ pairs from R can be routed in G_C.

3.2 Creating and Using the OS Instance

We now create a fractionally feasible OS instance on G_C, that is, an instance of EDP where the terminals are on C and we have a feasible fractional (partial) routing for demands defined over the terminals. Specifically, we create a set S of pairs of nodes chosen from C such that for each pair $uv \in S$, we have a flow $d(uv) \in [0, 1]$, and we may simultaneously route these flows while obeying the edge capacities in G_C. Moreover, we ensure that $\sum_{uv \in S} d(uv) = \Omega(p)$; recall that p is the total original flow on paths entirely contained in G_C, and R is the set of demand pairs involved in these flow paths. The pairs in S will serve as *surrogate pairs* for those in R, and we set up a mapping $\pi : R \to S$ with several properties that we describe later. In Section 3.3, we show that given a feasible multicommodity flow for an OS instance, there is a half-integral flow with only a constant factor loss in value. Thus, we may find a subset $S' \subset S$ such that $|S'| = \Omega(p)$ and pairs in S' can be routed integrally with congestion 2.

The idea is then to find a routable subset $R' \subset R$ by using S' and the mapping π. We do this as follows. For each $uv \in S'$ we add any pair $st \in \pi^{-1}(uv)$ to R'. To argue that R' is routable in

G_C, we exhibit a path collection $\mathcal{P}$ such that the endpoints of each pair $st \in R'$ are joined to the endpoints of $\pi(st) \in S'$. Further, no edge of the graph is contained in more than two of the paths in $\mathcal{P}$. This requires that we set up the OS instance, S and the mapping π in a careful fashion. The above description shows that we obtain a set $R' \subset R$ such that $|R'| = \Omega(p)$ and R' is routable in G_C with congestion 4. We now describe details of the construction of the OS instance.

Recall from Section 3.1 that Q is the set of terminals in G_C whose mate is also in G_C. For a set of nodes $A \subseteq V(G_C)$ let $b(A) = \sum_{v \in A \cap Q} b_v$ denote the total b value of terminals in A. We assume that $p \geq 10$ for otherwise we can route one pair and obtain a constant factor approximation. Our first goal is to partition the graph G_C into node-disjoint connected induced graphs $G[V_1], G[V_2], \ldots, G[V_h]$ such that for $1 \leq i \leq h$, $10 \leq b(V_i) \leq 10\Delta$. We do this by partitioning via a simple greedy procedure, described next. This ensures that $h \geq \lfloor 2p/(10\Delta) \rfloor$. Here Δ is the maximum degree of the graph. In our instances $\Delta \leq 4$, however it is easier to understand the description with Δ in place of the constant 4.

Find a spanning tree T of G_C and root it at some node r. If $b(V) \leq 10\Delta$ stop and output $V_1 = V(T)$. Otherwise, find the deepest node v in T such that the subtree of T rooted at v, T_v, has total b-weight at least 10. We set $V_1 = V(T_v)$ and remove it from T. Note that $b(V(T_v)) < 10(\Delta - 1) + 1 \leq 10\Delta$ if $v \neq r$. We continue the process by finding another deepest node in the remaining tree and remove it and so on. At the end if we have a tree with total weight less than 10 we simply add it to the tree found in the previous step. This clustering step is similar to that used in [5].

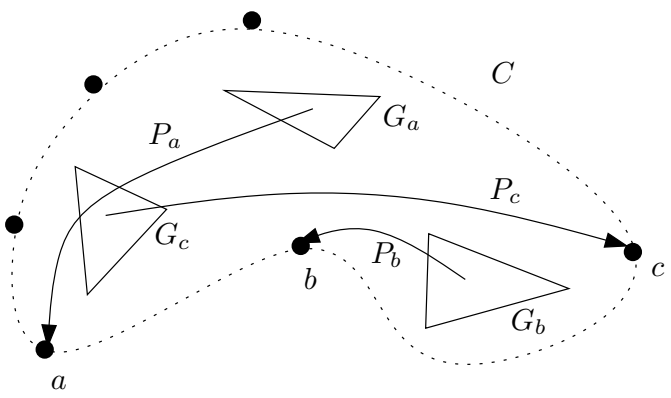

Figure 4: Setting up the OS instance.

Since every terminal v can simultaneously send $b(v)/10$ flow to the contour C, the node sets $V_1, V_2, \ldots, V_h$ can simultaneously send one unit of flow each to the contour C. It follows easily from the integrality of single source flows that there are edge-disjoint paths $P_1, P_2, \ldots, P_h$ where P_i has one end point on C and the other in V_i and further each node in C is the endpoint of at most one of these paths. To see this, one may consider a new "sink" node t adjacent to each node of C. We thus have a single-sink multicommodity flow to node t; we may actually orient the edges of G_C so that this flow induces a directed flow on an acyclic directed subgraph. In this directed graph, consider adding a new node s_i with directed arcs to the nodes of V_i and finally add a source node s with unit capacity arcs to each s_i. In the resulting directed acyclic graph we have a flow of size h from s to t, and the claim follows.

Let $C' \subseteq C$ be the collection of endpoints of the paths in $\mathcal{P} = \{P_1, P_2, \ldots, P_h\}$. By construction, we have a bijection $g : C' \to \{1, 2, \ldots, h\}$. For a node $a \in C'$ we refer to $G[V_{g(a)}]$, for simplicity of notation, as simply G_a and the path $P_{g(a)}$ by P_a. See Figure 3.2 for an illustration.

Now we are ready to define S and set up the mapping π from R to S. First, for any terminal

$s \in G_u$ we define $\pi(s) = u$. This map from Q to C' induces the desired map from R to pairs of nodes on C': for a pair $st \in R$ with $s \in G_u, t \in G_v$, we set $\pi(st) = uv$. We now define a feasible multicommodity flow between distinct node pairs u, v on C'.

For distinct $u, v \in C'$, we define

$$d(uv) = \frac{1}{3 \cdot 10 \cdot \Delta} \sum_{st \in R, s \in G_u, t \in G_v} b_{st}$$

and we add uv to S if $d(uv) > 0$. Note that for each $u \in C'$, $D(u) = \sum_{v \in C'} d(uv) \leq 1/3$. This follows from the fact that $b(V_i) \leq 10 \cdot \Delta$ for $1 \leq i \leq h$.

Lemma 3.2 *The multicommodity flow instance on S with demands d is feasible in G_C.*

Proof. Consider three identical copies of G_C, say H_1, H_2, H_3. The capacity of an edge in H_i is one-third of the capacity of its corresponding edge in G_C. We argue for the feasibility of d by considering some node $u \in C'$ and how it distributes its flow. The node u sends $D(u) \leq \frac{1}{3}$ flow along path P_u to the copy of G_u in H_1. We note that the path collection $\mathcal{P}$ is edge-disjoint in G_C and hence these flows are feasible in H_1 simultaneously for all nodes in C'. Let u' be the other endpoint of P_u. The node u' distributes $D(u)$ flow to the terminals in the copy of G_u in H_2. A terminal $s \in G_u$ is sent $\frac{1}{3 \cdot 10 \cdot \Delta} b_s$ flow. This can be done since G_u is connected and $\sum_{s \in G_u} b_s \leq 10\Delta$. Since the G_u's are disjoint, this distribution can be done simultaneously in H_2 for all such u' nodes. Now, the pairs in R use H_3 to route their multicommodity flow. This is feasible since the pairs had a feasible multicommodity flow in G_C and the flow is now scaled down by a factor of 3 at least. Composing these flows shows that d is feasible. □

Lemma 3.3 *Let $S' \subseteq S$ induce a matching on C'. Then there is a $R' \subset R$ and a path collection $\mathcal{Q}$ with the following properties: (i) $|R'| = |S'|$ and $\pi(R') = S'$, (ii) for each pair $st \in R'$, there are two paths $\mathcal{Q}_s$ and $\mathcal{Q}_t$ in $\mathcal{Q}$ that originate at s and t respectively and end at $\pi(s)$ and $\pi(t)$ respectively, and (iii) an edge e in G_C is in at most two paths in $\mathcal{Q}$.*

Proof. We create R' from S' as follows. For each $uv \in S'$ we pick an arbitrary but a single pair st such that $\pi(s) = u, \pi(t) = v$, and add st to R'. Such a pair must exist since $d(uv) > 0$. Therefore $|R'| = |S'|$ and $\pi(R') = S'$ as required. By construction of S we have that $s \in G_u$ and $t \in G_v$. We now create the paths $\mathcal{Q}_s$ and $\mathcal{Q}_t$. Let u' and v' be the endpoints of P_u and P_v in G_u and G_v respectively. Then $\mathcal{Q}_s$ is the path (maybe non-simple) obtained by composing a path from s to u' in G_u and the path P_u. Similarly, for $\mathcal{Q}_t$. We observe that the path collection $\mathcal{P}$ is edge-disjoint and the graphs $G_1, \ldots, G_h$ are node-disjoint and hence an edge can participate at most twice in the path collection $\mathcal{Q}$ created as above. □

Although the OS instance on S is well defined, it could be the case that $\sum_{uv \in S} d(uv)$ is significantly smaller than p, the total flow that we started with in G_C. We argue that this is the easy case.

Lemma 3.4 *If $\sum_{uv \in S} d(uv) \leq \frac{1}{6 \cdot 10 \cdot \Delta} p$, then $\Omega(p)$ pairs from R can be routed by edge-disjoint paths in G_C.*

Proof. We call a pair $st \in R$ to be *distant* if $s \in V_i$ and $t \in V_j$ and $i \neq j$. Let A be the set of distant pairs. Call a set V_i *good* if there is a pair $st \in R$ with $s, t \in V_i$. By construction we have that

$$\sum_{uv \in S} d(uv) = \frac{1}{3 \cdot 10 \cdot \Delta} \sum_{st \in A} b_{st}.$$

Therefore, if $\sum_{uv \in S} d(uv) \leq \frac{1}{6 \cdot 10 \cdot \Delta} p$, a large fraction of the pairs in R are not distant. It follows that the total number of good sets in this case is $\Omega(p/(10 \cdot \Delta))$. We can route one pair in each good set and the sets are node-disjoint and hence the routed pairs use edge-disjoint paths. □

3.3 Choosing the Demands in an OS instance

In this section we consider OS instances: instances in which the terminals are on the outer face of a planar graph. In such cases, the well-known Okamura-Seymour theorem states that the cut condition is sufficient to assure the existence of a half-integral flow. Here we are interested in the maximization version where not all pairs may be simultaneously routable. Even on trees (a very restricted OS instance) it is known that the maximization problem is APX-hard [14]. We solve the LP for the problem which gives an upper bound and then show that we can recover a constant fraction of the LP solution value. We make some preliminary observations that lead us to set up the problem as an abstract ring routing problem. We assume that the given graph is two-node connected and hence the outer face can be assumed to be a connected ring. A reduction to the two-node-connected case is presented in Section 3.4.

For OS instances one can focus on a restricted set of cuts determined by pairs of edges on the outer face. Let e and e' be two edges on the outer face. Consider the cheapest cut in the graph that contains e and e' and let its value be $\mu(e, e')$: this can be obtained by a shortest path computation in the dual of the embedded graph. For an OS instance to be feasible, it suffices to check that the cut condition holds only for the above types of cuts ([25], cf. [30]). Using this, we set up our problem as a ring-routing problem below.

In the following, let $G = (V = \{0, 1, 2 \ldots, n-1\}, E)$ be an undirected ring with edges $e_i = i(i+1)$ for $0 \leq i < n$. We assume that the nodes are numbered clockwise around the cycle. In addition, we assume that for each pair of edges e_i, e_j, there is an associated integer capacity $\mu(e_i, e_j)$, or simply $\mu(ij)$. We also consider a *demand multiset* F of edges with endpoints in V. Our sets may also be *weighted* in that for each $f \in F$ there is a nonnegative demand value $d_f \in [0, 1]$. The weight of such a set is then $d(F) = \sum_{f \in F} d_f$. An edge f *crosses* a pair e_i, e_j if the endpoints of f lie in distinct components of $G - \{e_i, e_j\}$. The *load* on a pair (cut) e_i, e_j, denoted by $l(ij)$, is just the sum of the demands that cross the pair, i.e., $\sum_{f \in F(ij)} d_f$, where $F(ij)$ are those demands that cross e_i, e_j. A weighted set of demands is *feasible* (*almost-feasible*) if for each e_i, e_j we have that its load is at most $\mu(e_i, e_j)$ (resp. $\mu(e_i, e_j) + 2$).

Our main result in this section is that if we start with a feasible fractional set of demands, then we may find a large integral set of demands that obey the "abstract" 2-cut conditions imposed by μ. A similar problem (the Pairing Lemma) was solved by Frank [12]. In his setting, Frank gave necessary and sufficient conditions for the existence of a perfect matching on V that obeys the μ-cut condition. A key difference is that Frank works with the complete graph, that is, his matching may use any demand edge ij, while we may select only the demands from F.

Theorem 3.5 (Abstract Capacity Ring Lemma) *For any abstract capacity ring $G, \mu : E \times E \rightarrow \mathbf{Z}_+$ and feasible weighted set of demands F, d, there is an almost-feasible subset F' such that each demand in F' is a unit demand, and $|F'| \geq \alpha \sum_{f \in F} d_f$ for some fixed constant $\alpha > 0$.*

The rest of the section is devoted to the proof of the above theorem. For a node $v \in V$ we let $w(v) = \sum_{f : v \in f} d_f$ denote the total weight of demand edges incident to v. For a set of nodes S we let $w(S) = \sum_{v \in S} w(v)$. We ensure that in the final solution the number of demands incident to any node of V is no more than $\lceil w(v) \rceil$. We assume without loss of generality that $w(v) \leq 1$. Otherwise we can split a node into multiple nodes connected by a path of infinite capacity edges.

In the following we use γ for a fixed integer that is at least 6. A *block* in our instance is a contiguous interval of nodes S on the ring such that $\gamma \leq w(S) < 2\gamma + 1$. At any given time we are working with a partition of the nodes into blocks $B_1, B_2, \ldots, B_h$ and so $h = \Omega(\sum_f d_f)$. Also, for a node u, we denote by $B(u)$ the block containing u. If the total demand is less than γ, then picking any single demand produces the desired subset F'. Otherwise, we start with a feasible blocking obtained by a simple greedy procedure that goes clockwise around the ring picking minimal blocks to satisfy the requirement that $\gamma \leq w(B_i) < 2\gamma + 1$. The blocks we create are intervals on the ring and there is a natural notion of adjacency between them. As our algorithm proceeds, the blocks will change. At all times, a block maintains one of the two labels: *locked* or *live*. Initially, all blocks are labeled live.

We proceed to generate our feasible set of demands by repeating three operations: *augmenting*, *locking* and *reblocking*. An augmentation of a demand edge f involves pushing up its value d_f to 1; at the same time we sacrifice an $O(\gamma)$ amount of other demands to relieve the load on cuts that f crosses. The demand $f = uv$ is then selected for F' and $B(u), B(v)$ are locked, and all demands with an endpoint in $B(u)$ or $B(v)$ are deleted. Once a block is locked no future blocks can contain any node from it.

Reblocking: We first describe the reblocking procedure which is used after any operation which deletes edges from our instance, and thus may produce blocks that are too light, i.e., of weight less than γ. Suppose that B is such a block and let B_l, B_r be its neighboring blocks. If either B_l, B_r is not yet locked, say B_l, then we merge B and B_l. If the weight of this merged block is more than $2\gamma + 1$, then we may *split* only part of B_l off to merge with B, thus creating two new blocks. Note that this does not affect the weight of any of the other blocks. If both B_l and B_r are *locked*, then we delete the edges incident to B and charge them to B_l, B_r. We do this so that each of B_l, B_r gets charged the same amount. It is best to view this as splitting B in two parts which are then merged with B_l, B_r respectively to make these locked blocks a little larger. (We sometimes refer to the new blocks B'_l, B'_r as the *expanded blocks*.) Note that each locked block could get charged this way by at most one block on either side, and since every locked block contributes a routed demand to F', we will be able to establish that at most a constant fraction of demand is lost in this way. A block that is not locked is called a *live* block. In handling B, other light blocks may be created when we delete the demands incident to B. We repeat this process until all remaining live blocks are again of weight at least γ and at most $2\gamma + 1$.

Augmenting: We next describe the augmenting procedures. Each of these procedures consists of carefully choosing a demand edge f and increasing d_f to 1 and adding it to F'. The blocks that contain the end points of f (could be a single block or a pair of blocks) are then locked and all demand edges incident to those blocks are deleted (as we will see, sometimes we delete edges incident to multiple blocks). After adding an edge to f, we always perform reblocking as necessary. We repeat these augmentations until we have no more live blocks left. We now describe how to pick a demand f.

First, if there is some block B with a demand edge f with both ends in B, then we pick it. If not, we check if there are two distinct blocks B and B' such that the total weight of demands between them is at least 1. If so, we pick an arbitrary demand f between them. Call these two augmentations *simple*, of *type I* and *II* respectively. Note that if no simple augmentations exist, then each block B has demand edges to at least γ other distinct blocks.

We next describe a more involved augmenting procedure called a *complex* augmentation. The intuition behind the procedure is based on the following operation. Consider any pair of demand edges uv, vw where say $u < v < w$. If u has a neighbour z in the interval $[v, w]$, then a *cross operation* is where we increase the demand on uv by some ϵ and reduce uz and vw by ϵ. This does not increase the load on any cut since any cut crossed by uv is either crossed by uz or vw. We look for several simultaneous cross operations for uv that can be used together to increase its demand all the way to

1, and thus allow us to select uv for F'. Identifying a demand edge uv for which this is possible is the key.

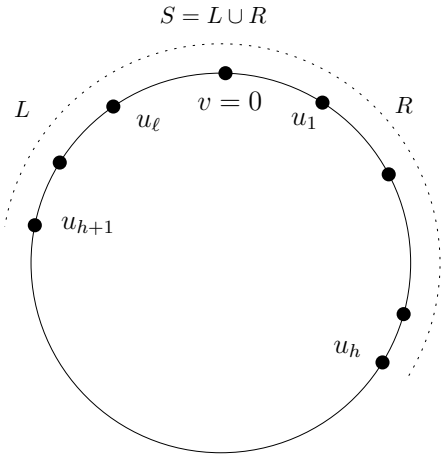

Figure 5: A cover for node v in the graph G'.

Let $B_0, B_1, \ldots, B_{n'-1}$ be the current set of live blocks (not locked). We create an undirected ring $G' = (V' = \{0, 1, \ldots, n'-1\}, E')$ with the nodes corresponding to these blocks; the nodes are numbered clockwise. Let $H = (V', X)$ be a demand graph induced on the blocks where $uv \in X$ if there is some demand edge f incident to nodes in B_u and B_v. We also associate weights with edges in X. We abuse notation and use d for this. The weight of uv, $d(uv)$ is the total sum of weights of demand edges remaining in F with one endpoint in B_u and the other in B_v. If no simple augmentation is feasible then $d(uv) < 1$ and the degree of a node in H is at least $\gamma + 1$. We now work on the ring G'.

We say a contiguous segment A of the ring "covers" node a if it contains a and $\sum_{u \in A} d(ua) \geq 4$. Choose a node v with a cover S of minimum length (in terms of nodes). Without loss of generality, assume $v = 0$ and v's neighbours on the segment S are $u_1 < u_2, \ldots < u_\ell$, so subscripts are numbered in clockwise direction - see Figure 5 where only the nodes u_i are depicted. The segment itself is the union of a "left" segment $L = [u_{h+1}, 0]$ and a right one $R = [0, u_h]$, i.e., S is the segment $[u_{h+1}, u_{h+1}+1, \ldots, u_l \ldots v = 0, 1, 2, \ldots, u_h]$. See Figure 5. Without loss of generality, $\sum_{a \in R} d(av) \geq 2$. Since we had no simple augmentations available, we may deduce that

$$\sum_{k=2}^{h} d(vu_k) > 1 \text{ and } h \geq 3. \tag{2}$$

Let f be an arbitrary demand edge in G between blocks B_v and B_{u_1}. We choose f for augmentation. This entails that we lock block B_v and B_{u_1} and delete all edges incident to them other than f. We argue that augmenting f is a safe operation because, by deleting all edges incident to B_v and B_{u_1}, we could apply several cross operations that push f's demand to 1. These operations involved f, and pairs of edges e, e' where e has one end in $B_v = B_0$ and the other in some $B_{u_k}, k = 2, \ldots, h$ and e' has one end in B_{u_1} and the other end in $B_{k'}$ for some $k' \in [u_h + 1, n' - 1] = \{u_h + 1, u_h + 2, \ldots, n' - 1\}$. Equation (2) assures us that there is enough weight on the first "e-type" edges to be able to increase f's demand to 1. The fact that there is enough weight on these "e'-type" edges is assured by the following.

Claim 3.6 *For $\gamma \geq 6$ we have $\sum_{y=u_h+1}^{n'-1} d(u_1 y) \geq 1$.*

Proof. At this moment, since B_{u_1} is a live block, the total demand incident to u_1 is at least $\gamma \geq 6$. If $\sum_{y=u_h}^{n'-1} d(u_1 y) < 2$, then the segment $[0, u_h - 1]$ would be a cover for u_1 since the demand incident

to it is at least $\gamma - 2 \geq 4$. This would contradict the choice of S as a minimum length cover; so $\sum_{y=u_h}^{n'-1} d(u_1 y) \geq 2$. Since we had no simple augmentation, $d(u_1 u_h) < 1$, and hence $\sum_{y=u_h+1}^{n'-1} d(u_1 y) \geq 1$ as required. □

This finishes the description of the augmenting procedures and the algorithm. Let F' be the set of demands chosen during the course of the algorithm. We prove two properties of F'.

Lemma 3.7 *There is a constant α, dependent only on γ, such that $|F'| \geq \alpha \sum_{f \in F} d_f$.*

Proof. We have argued that an augmentation can be done as long as there are live blocks, therefore at the end of the algorithm there are no live blocks left. We show the lemma by a charging argument that the total demand lost during the algorithm is not more than a constant times $|F'|$. Note that we lose demand for two reasons: (i) when we delete demand edges after an augmentation and (ii) when we apply reblocking operations. Each augmentation results in a demand f being added to F'. Let B_1 and B_2 be the blocks that contained the endpoints of f (in case f has endpoints in the same block then $B_1 = B_2$). These blocks are locked and f is then associated to these blocks B_1 and B_2. All demand edges incident to those blocks are then deleted and we charge the weight of the deleted demands to f. Recall that the weight of a block is at most $2\gamma + 1$, and so f is charged at most $4\gamma + 2$ in this way. During reblocking operations we might have taken a light block B and charged and merged it to previously locked blocks, such as B_1. However, as we argued, at most two such blocks can be charged to B_1, one from its "left" and one from its "right". Also, any such light block charged half its weight to the live block (i.e., to the demand associated with that block) on either side of it. Since each light block is incident to at most a demand of γ, a locked block gets charged in this way at most γ, and hence a demand (which is associated with 2 blocks) would get charged at most 2γ. Combining all of the charges we see that $|F'| \geq \frac{1}{6\gamma+3} \sum_{f \in F} d_f$. □

Lemma 3.8 *The set F' is an almost-feasible set of demands for G, μ. In fact, $|F'(ij)| \leq 2 + \lfloor l(ij) \rfloor$ for each $i, j \in \{0, 1, \ldots, n-1\}$.*

Proof. Let e_i and e_j be two edges of the ring. Recall that $F(i,j)$ is the set of demands that cross the cut induced by e_i, e_j, and that $l(ij)$ denotes the load of F on the cut induced by e_i, e_j, i.e., $\sum_{f \in F(ij)} d_f$. By the feasibility of F we have that $l(ij) \leq \mu(ij)$ for all i, j. We say that an edge e_i belongs to a locked block B if both endpoints of e_i lie in B. Let $z_i = 1$ if e_i lies in a locked block and 0 otherwise. If $z_i = 1$ we let f_i be the unique edge in F' associated with the locked block containing e_i. Let $A(ij)$ be the set of demands in $F'(ij)$ after removing the edges f_i and f_j if they exist. We show that $|A(ij)| \leq \lfloor l(ij) \rfloor$. Since $|F'(ij)| \leq |A(ij)| + 2$, the result will follow.

Consider an edge f in $A(ij)$ and note that f has its endpoints in distinct locked blocks when the algorithm terminates. We consider the possible types of augmentations that resulted in f being added to F'. First, f could not have been added in a simple type I augmentation. In that case, f would have had both ends in some block B when it was added. Since B was then locked, and locked blocks are never split (but possibly expanded), f would never have its ends in distinct blocks. In the following, let B_1 and B_2 be the two blocks that existed when f was added to F'. Suppose next that f was added in a type II simple augmentation. Since e_i and e_j do not belong to either B_1 or B_2 (otherwise f would not be in $A(ij)$), all demands that were incident to both B_1 and B_2 crossed e_i and e_j when f was added. The total weight of such demands between B_1 and B_2 was at least 1 when we added f and deleted those demands. Therefore the load of f on this cut could be charged to those deleted demands.

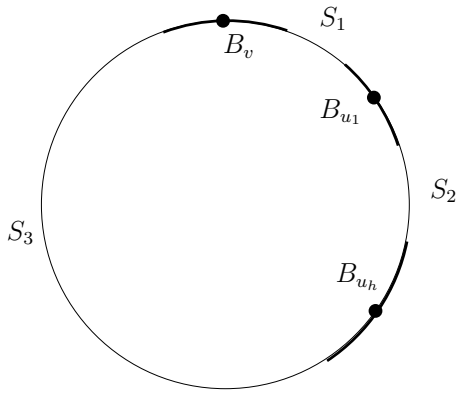

Figure 6: Analysis of augmentation.

Finally, suppose that f was added in a complex augmentation. In this case the weight of demands between B_1 and B_2 can be small and we need to use the careful choice of the demand f. We refer to the notation used when describing this type of augmentation. We have that f is an edge between block B_v and B_{u_1}. We consider three segments, S_1, S_2, S_3, of the ring G obtained by removing the edges in the blocks B_v, B_{u_1} and B_{u_h}. S_1 is the segment "between v and u_1", S_2 is the segment between u_1 and u_h and S_3 is the segment between u_h and v. See Figure 6. Note that e_i, e_j must lie either in B_{u_h} or one of S_1, S_2, S_3 or else f would not be in $A(ij)$. Since f crosses the cut for e_i, e_j we assume without loss of generality that e_i is in the segment S_1. We consider two cases based on the location of e_j. If e_j is in S_2 or B_{u_h} we charge f's load on the $\{e_i, e_j\}$-cut to the loads from demand edges from B_{u_1} to the segment S_3. Recall from Claim 3.6 that there was indeed at least a demand of 1 amongst these edges. Also note that all these edges are removed when B_{u_1} is locked after choosing f. Similarly, if e_j is in S_3 we charge f's load to the load of edges from B_v to S_2 and B_{u_h}. Recall also that this was possible since the demand amongst these edges was at least 1 by (2). □

Lemmas 3.7 and 3.8 finish the proof of Theorem 3.5.

Now we come back to OS instances. Recall that we assumed that the graph is two-node connected and hence the outer face is a cycle. From this it follows that $\mu(ij) \geq 2$ for any two edges e_i and e_j on the ring formed by the outer face. From Lemma 3.8 we see that the set of demands in the final solution F' satisfies the condition that $|F'(ij)| \leq 2 + \lfloor l(ij) \rfloor$. By initially scaling down the demand values, we can assume that $l(ij) < \mu(ij)/2$. Therefore it follows that $2 + \lfloor l(ij) \rfloor \leq \mu(ij)$ and hence $|F'(ij)| \leq \mu(ij)$. We thus obtain that for OS instances on two-node connected planar graphs, the integrality gap for the cardinality version of the all-or-nothing multicommodity flow problem is $O(1)$. This together with the reduction from the next section yields the following theorem.

Theorem 3.9 *For OS instances in planar graphs, the integrality gap for the cardinality version of the all-or-nothing multicommodity flow problem is* $O(1)$.

3.4 Reduction to the Two-Node Connected Case

Given an instance of EDP or ANF, we now show that it can be reduced to a collection of instances in which each graph is two-node-connected. The formal statement of the reduction is given below.

Theorem 3.10 *Let G be a multi-graph of maximum degree Δ and let $(G, \mathcal{T})$ be an instance of EDP (or ANF) with a feasible multicommodity flow $\bar{f}$. Let $G_1, G_2, \ldots, G_h$ be the two-node connected components*

of G and $\mathcal{T}_i \subseteq \mathcal{T}$ be the demand pairs with both end points in G_i, and let $\bar{f}_i$ denote the flow induced by $\bar{f}$ restricted to the instance $(G_i, \mathcal{T}_i)$. Then, either $\sum_{i=1}^{h} val(\bar{f}_i) \geq val(\bar{f})/2$ or there is a polynomial time algorithm that routes $\lceil val(\bar{f})/(2\Delta) \rceil$ pairs.

We remark that in the above theorem, Δ can be replaced by an absolute constant using a more involved proof. Since we work with graphs with $\Delta = 4$ in this paper, we give only the simpler proof.

Proof. We describe a proof for EDP; the proof is identical for ANF. We assume without loss of generality that G is connected. To prove the theorem we consider the case that $\sum_{i=1}^{h} val(\bar{f}_i) < val(\bar{f})/2$. Remove all pairs uv from $\mathcal{T}$ with u, v in the same G_i for some $1 \leq i \leq h$. Let $\mathcal{T}'$ be the pairs that remain. By our assumption, the total flow for the pairs in $\mathcal{T}'$ is at least $val(\bar{f})/2$. We now describe a polynomial time algorithm that routes $\lceil val(\bar{f})/(2\Delta) \rceil$ pairs from $\mathcal{T}'$.

Consider the block-cut tree T of G. The nodes of T are the *blocks* (maximal two-node connected subgraphs) of G and the *cut-nodes* of G. An edge connects a block B and cut-node v if the block B contains v. We root T at an arbitrary block-node r. For any node z of T, we let G_z denote the subgraph of G that corresponds to node z: it is a singleton node if z is a cut-node and is a two-node connected graph if z is a block-node. Furthermore, we denote by G^*_z, the subgraph of G induced by the union of the nodes of G contained in all the blocks in the subtree of T rooted at z. Our algorithm will crucially use the property that every path that connects a pair in $\mathcal{T}'$ goes through one or more cut-nodes in T and at least two blocks. We process the tree T and G as follows. Let y be a cut-node in T that is farthest from r (a deepest node) such that G^*_y contains both end points of some demand pair in $\mathcal{T}'$. We observe the following. Note that by our choice of y, for any demand pair $uv \in \mathcal{T}'$ with either u or v in G^*_y, every path connecting u and v in G goes through y.

Let $st \in \mathcal{T}'$ be an arbitrary demand pair with both endpoints in G^*_y. The algorithm connects (routes) the pair st in G^*_y and removes all pairs in $\mathcal{T}'$ with an end point in G^*_y. From the above observation, the total flow lost in removing these pairs is at most Δ since any flow for such pairs goes through y and the degree of y is at most Δ. We charge the lost flow to the routed demand pair st. Note that no remaining pair needs to use any edge in G^*_y. Hence we remove these edges from G and recurse on the demand pairs remaining in $\mathcal{T}'$. It is easy to see that the algorithm thus routes $\lceil val(\bar{f})/2\Delta \rceil$ pairs. □

4 Well-Linked Decompositions and Recursively Divisible Planar Graphs

We close by briefly discussing lower bounds on obtaining approximations for EDP and ANF via the approach of well-linked decompositions. For us a well-linked decomposition is a partition of the graph into node disjoint induced subgraphs such that each induced subgraph retains a collection of well-linked terminals. We say a terminal is *retained* in the decomposition if it is part of the specified (flow)-well-linked set in one of the subgraphs. We are interested in the ratio of the total number (or more generally, total fractional weight) of terminals retained as a fraction of OPT, the value of an optimum solution to the flow relaxation. Formal definitions involve the notions of both flow and cut-well-linkedness, and the reader is referred to [7, 9] for more details. For our present purposes we repeat only the general bounds known for general graphs. In [7] it is shown that in general graphs one can retain $\Omega(1/(\beta(G) \log n))$ of the flow by recursive cutting along sparse cuts ($\beta(G)$ is the best approximation ratio for finding sparse cuts). In [9], a class of instances is shown for which it is not possible to retain better than $O(\text{OPT}/\log(n))$. In planar graphs, we do not have a formal result but we now describe a family of instances for which we believe one cannot hope to retain a constant fraction of OPT.

Related to our comments in Section 1.3 we describe a class of planar graphs that are bad for a recursive sparse-cut algorithm if the partition procedure uses a particular sequence of ϵ-sparse cuts for some fixed constant $\epsilon > 0$. These graphs may well have the stronger property that no well-linked decomposition is possible that retains a constant fraction of the flow. Although we do not have concrete evidence for this, (the structure of the graphs is still a bit mysterious and fascinating to us), we think that these graphs are suggestive of a lower bound.

We now define this class of recursively divisible planar graph instances denoted by P_n. For any constant ϵ, these are graphs such that if we repeatedly split along ϵ-sparse cuts, then we may end up with a collection of subgraphs that retain very little of the initial flow. It is easier to view how these graphs arise by looking at their planar dual, denoted by D_n. We suppose at first that $\epsilon = 1$. Each edge of D_n has an associated (not Euclidean) length $l(e)$. The graph D_1 can be thought of as a circle C of length 1 with 2 nodes u, v at antipodal points on the circle. There is also another edge of length $1/2$ joining u, v. We now recursively define D_{n+1} from D_n as follows. At any point, D_n is outerplanar and the boundary of each face F can be written as $C_F \cup I_F$, where C_F is a segment of the original cycle C, and I_F is a curve inside C. Let i_F be the midpoint (with respect to the length function l) of the curve I_F (note that there may be no node at this midpoint). Let c_F denote the midpoint of C_F. In D_{n+1}, we add the nodes i_F, c_F, and join them by an edge of length $\frac{1}{2^n}$. In Figure 1.3, the graphs (without lengths) D_6, D_8, D_{10} are shown.

The recursively divisible graph P_n is obtained by drawing the dual of D_n and removing the node corresponding to the infinite face and its incident edges. For an edge $uv \in D_n$, it contributes $2^n l(uv)$ parallel edges in P_n. With each edge in P_n we associate a demand pair in the routing instance and hence the total flow routed is the same as the total number of edges in P_n. The recursive definition of D_n gives rise naturally to a sparse cutting procedure in P_n. At the i^{th} level, there are 2^i faces in D_n, each containing 2^{n-i} nodes of P_n. Each such face is then split in half and the two halves are joined by 2^{n-i} edges. We may ensure ϵ-sparseness by adding $1/\epsilon$ terminals at each node in P_n. Thus for any fixed ϵ, a recursive procedure that partitions along these ϵ-sparse cuts, removes almost all the edges of P_n and hence the associated demands.

5 Concluding Remarks

We believe that an $O(1)$ approximation for MEDP in planar graphs can be obtained with congestion 2 instead of 4. It seems feasible to extend our approach to graphs with bounded genus and to graphs that exclude a fixed minor.

Acknowledgments: Sanjeev Khanna is supported in part by an NSF Career Award CCR-0093117 and by NSF Award CCF-0635084. This work was done while the first author was at Lucent Bell Labs. Chandra Chekuri and F. Bruce Shepherd acknowledge support from an ONR basic research grant N00014-05-1-0256 to Bell Labs.

References

[1] M. Andrews and L. Zhang. Hardness of the undirected edge-disjoint paths problem. *Proc. of ACM STOC*, 276–283, 2005.

[2] M. Andrews, J. Chuzhoy, S. Khanna and L. Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. *Proc. of IEEE FOCS*, 226–244, 2005.

[3] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. on Comp.*, 27(1):291–301, 1998.

[4] C. Chekuri, M. Mydlarz, and F. B. Shepherd. Multicommodity Demand Flow in a Tree and Packing Integer Programs. To appear in *ACM Transactions on Algorithms*. Preliminary version in *Proc. of ICALP*, 2003.

[5] C. Chekuri, S. Khanna, and F. B. Shepherd. The All-or-Nothing Multicommodity Flow Problem. *Proc. of ACM STOC*, 156–165, 2004.

[6] C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-Disjoint Paths in Planar Graphs. *Proc. of IEEE FOCS*, 71–80, 2004.

[7] C. Chekuri, S. Khanna, and F. B. Shepherd. Multicommodity Flow, Well-linked Terminals, and Routing Problems. *Proc. of ACM STOC*, 183–192, 2005.

[8] C. Chekuri, S. Khanna, and F. B. Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, Vol 2, 137–146, 2006.

[9] C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-disjoint paths in planar graphs with constant congestion. *Proc. of ACM STOC*, 757–766, 2006.

[10] S. Even, A. Itai and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. on Computing*, Vol. 5, 691-703, 1976.

[11] L. R. Ford, D. R. Fulkerson. Flows in Networks. Princeton University Press, Princeton, NJ, 1962.

[12] A. Frank. Edge-disjoint paths in planar graphs. *J. of Combinatorial Theory, Ser. B.*, No. 2, 164–178, 1985.

[13] A. Frank. Packing paths, cuts, and circuits - a survey. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, eds., *Paths, Flows and VLSI-Layout*, 49–100. Springer Verlag, Berlin, 1990.

[14] N. Garg, V. Vazirani, M. Yannakakis. Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees. *Algorithmica*, 18(1):3–20, 1997.

[15] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher, Eds., New York: Plenum Press, 85–103, 1972.

[16] P. Klein, S. Plotkin and S. Rao. Planar graphs, multicommodity flow, and network decomposition. *Proc. of ACM STOC*, 682–690, 1993.

[17] J. M. Kleinberg. Approximation algorithms for disjoint paths problems. PhD thesis, MIT, May 1996.

[18] J. M. Kleinberg. An Approximation Algorithm for the Disjoint Paths Problem in Even-Degree Planar Graphs. *Proc. of IEEE FOCS*, 627–636, 2005.

[19] J. M. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. *JCSS*, 57:61–73, 1998.

[20] J. M. Kleinberg and É. Tardos. Disjoint Paths in Densely Embedded Graphs. *Proc. of IEEE FOCS*, 52–61, 1995.

[21] S. G. Kolliopoulos. Edge Disjoint Paths and Unsplittable Flow. *Handbook on Approximation Algorithms*, Chapman Hall/CRC Press, to appear.

[22] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *JACM*, 46(6):787–832, 1999. Prelim. version in *Proc. of IEEE FOCS*, 1988.

[23] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. Preliminary version in *Proc. of IEEE FOCS*, 1994.

[24] K. Menger. Zur Allgemeinen Kurventheorie Fundam. Math. **10**, 96–115, 1927.

[25] H. Okamura and P.D. Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31, 75–81, 1981.

[26] H. Räcke. Minimizing congestion in general networks. *Proc. of IEEE FOCS*, 43–52, 2002.

[27] S. Rao. Small distortion and volume preserving embeddings for planar and Euclidean metrics. *Proc. of ACM SoCG*, 300–306, 1999.

[28] N. Robertson and P. D. Seymour. Outline of a disjoint paths algorithm. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, Eds., *Paths, Flows and VLSI-Layout*. Springer-Verlag, Berlin, 1990.

[29] N. Robertson, P. D. Seymour and R. Thomas. Quickly Excluding a Planar Graph. *Journal of Combinatorial Theory (B)*, 62: 323-348, 1994.

[30] A. Schrijver. Combinatorial Optimization: Polyhedra and Efficiency. Springer-Verlag, 2003.