A Polylogarithimic Approximation Algorithm for Edge-Disjoint Paths with Congestion 2

Julia Chuzhoy *
Toyota Technological Institute,
Chicago, IL, USA
cjulia@ttic.edu

Shi Li †
Center for Computational Intractability
Department of Computer Science, Princeton University
Princeton, NJ, USA
shili@cs.princeton.edu

***Abstract*—In the Edge-Disjoint Paths with Congestion problem (EDPwC), we are given an undirected n-vertex graph G, a collection $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of demand pairs and an integer c. The goal is to connect the maximum possible number of the demand pairs by paths, so that the maximum edge congestion - the number of paths sharing any edge - is bounded by c. When the maximum allowed congestion is $c = 1$, this is the classical Edge-Disjoint Paths problem (EDP).**

The best current approximation algorithm for EDP achieves an $O(\sqrt{n})$-approximation, by rounding the standard multi-commodity flow relaxation of the problem. This matches the $\Omega(\sqrt{n})$ lower bound on the integrality gap of this relaxation. We show an $O(\mathrm{poly}\log k)$-approximation algorithm for EDPwC with congestion $c = 2$, by rounding the same multi-commodity flow relaxation. This gives the best possible congestion for a sub-polynomial approximation of EDPwC via this relaxation. Our results are also close to optimal in terms of the number of pairs routed, since EDPwC is known to be hard to approximate to within a factor of $\tilde{\Omega}\left((\log n)^{1/(c+1)}\right)$ for any constant congestion c. Prior to our work, the best approximation factor for EDPwC with congestion 2 was $O(n^{3/7})$, and the best algorithm achieving a polylogarithmic approximation required congestion 14.

***Keywords*-approximation algorithms; network routing; edge-disjoint paths**

I. Introduction

One of the central and most extensively studied graph routing problems is the Edge-Disjoint Paths problem (EDP). In this problem, we are given an undirected n-vertex graph $G = (V, E)$, and a collection $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of k source-sink pairs, that we also call demand pairs. The goal is to find a collection $\mathcal{P}$ of edge-disjoint paths, connecting the maximum possible number of the demand pairs.

Robertson and Seymour [1] have shown that EDP can be solved efficiently, when the number k of the demand pairs is bounded by a constant. However, for general values of k, it is NP-hard to even decide whether all pairs can be simultaneously routed via edge-disjoint paths [2]. A standard approach to designing approximation algorithms for EDP and other routing problems, is to first compute a multi-commodity flow relaxation, where instead of connecting the demand pairs with paths, we are only required to send the maximum amount of multi-commodity flow between the demand pairs, with at most one flow unit sent between every pair. Such a fractional solution can be computed efficiently by using the standard multi-commodity flow LP-relaxation, and it can then be rounded to obtain an integral solution. Indeed, the best current approximation algorithm for the EDP problem, due to Chekuri, Khanna and Shepherd [3], achieves an $O(\sqrt{n})$-approximation using this approach. Unfortunately, a simple example by Garg, Vazirani and Yannakakis [4], shows that the integrality gap of the multi-commodity flow relaxation can be as large as $\Omega(\sqrt{n})$, thus implying that the algorithm of [3] is essentially the best possible for EDP, when using this approach. This integrality gap appears to be a major barrier to obtaining better approximation algorithms for EDP. Indeed, we do not know how to design better approximation algorithms even for some seemingly simple special cases of planar graphs, called the brick-wall graphs. With the current best hardness of approximation factor standing on $\Omega(\log^{1/2-\epsilon} n)$ for any constant ϵ (unless NP is contained in $\mathsf{ZPTIME}(n^{\mathrm{poly}\log n})$ [5], [6]), the approximability of the EDP problem remains one of the central open problems in the area of routing.

A natural question is whether we can obtain better approximation algorithms by slightly relaxing the disjointness requirement, and allowing the paths to share edges. We say that a set $\mathcal{P}$ of paths is an α-approximate solution with congestion c, iff the paths in $\mathcal{P}$ connect at least OPT/α of the demand pairs, while every edge of G appears on at most c paths in $\mathcal{P}$. Here, OPT is the value of the optimal solution to EDP, where no congestion is allowed. This relaxation of the EDP problem is called EDP with congestion (EDPwC). The EDPwC problem is a natural framework to study the tradeoff between the number of pairs routed and the congestion, and it is useful in scenarios where

* Supported in part by NSF CAREER grant CCF-0844872 and Sloan Research Fellowship.
† Supported by NSF awards MSPA-MCS 0528414, CCF 0832797, AF 0916218 and CCF-0844872.

we can afford a small congestion on edges.

The classical randomized rounding technique of Raghavan and Thompson [7] gives a constant factor approximation for EDPwC, when the congestion c is $\Omega(\log n/\log\log n)$. More generally, for any congestion value c, factor $O(n^{1/c})$-approximation algorithms are known for EDPwC [8], [9], [10]. Recently, Andrews [11] has shown a randomized $O(\mathrm{poly}\log n)$-approximation algorithm with congestion $c = O(\mathrm{poly}\log\log n)$, and Chuzhoy [12] has shown a randomized $O(\mathrm{poly}\log k)$-approximation algorithm with congestion 14. For the congestion value $c = 2$, Kawarabayashi and Kobayashi [13] have recently shown an $\tilde{O}(n^{3/7})$-approximation algorithm, thus improving the best previously known $O(\sqrt{n})$-approximation for $c = 2$ [8], [9], [10]. We note that all the above mentioned algorithms rely on the standard multi-commodity flow LP relaxation of the problem. It is easy to see that the values of the optimal solution of this LP relaxation for the EDP problem, where no congestion is allowed, and for the EDPwC problem, where congestion c is allowed, are within a factor c from each other. Therefore, the statements of these results remain valid even when the approximation factor is computed with respect to the optimal solution to the EDPwC problem.

In this paper, we show a randomized $O(\mathrm{poly}\log k)$-approximation algorithm for EDPwC with congestion 2. Given an instance $(G, \mathcal{M})$ of the EDP problem, our algorithm w.h.p. routes at least $\Omega(\mathsf{OPT}/\mathrm{poly}\log k)$ pairs with congestion 2, where OPT is the maximum number of pairs that can be routed with no congestion. Our algorithm also achieves an $O(\mathrm{poly}\log k)$-approximation when compared with the optimal solution to EDPwC with congestion 2. The algorithm performs a rounding of the standard multi-commodity flow relaxation for EDP. Therefore, our result shows that when congestion 2 is allowed, the integrality gap of this relaxation improves from $\Omega(\sqrt{n})$ to polylogarithmic. Our result is essentially optimal with respect to this relaxation, both for the congestion and the number of pairs routed, in the following sense. As observed above, if we are interested in obtaining a sub-polynomial approximation for EDP via the multi-commodity flow relaxation, then the best congestion we can hope for is 2. On the other hand, Andrews et al. [6] have shown that the integrality gap of the multi-commodity flow relaxation for EDPwC is $\Omega\left(\left(\frac{\log n}{(\log\log n)^2}\right)^{1/(c+1)}\right)$ for any constant congestion c. In particular, the integrality gap for congestion 2 is polylogarithmic, though the degree of the logarithm is much lower than the degree we obtain in our approximation algorithm. Andrews et al. [6] have also shown that for any constant ϵ, for any $1 \leq c \leq O\left(\frac{\log\log n}{\log\log\log n}\right)$, there is no $O\left((\log n)^{\frac{1-\epsilon}{c+1}}\right)$-approximation algorithm for EDPwC with congestion c, unless NP $\subseteq$ ZPTIME$(n^{\mathrm{poly}\log n})$. In particular, this gives an $\Omega\left(\log^{(1-\epsilon)/3} n\right)$-hardness of approximation for EDPwC with congestion 2.

Our results. Our main result is summarized in the following theorem.

Theorem 1: There is an efficient randomized algorithm, that, given a graph G, and a collection $\mathcal{M}$ of k source-sink pairs, w.h.p. finds a routing of $\Omega(\mathsf{OPT}/(\mathrm{poly}\log k))$ of the pairs in $\mathcal{M}$ with congestion at most 2, where OPT is the maximum number of pairs that can be routed with congestion 2.

Organization. We start with preliminaries in Section II and provide an overview of our algorithm in Section III. We develop machinery to analyze vertex clusterings in Section IV, and complete the algorithm description in Sections V and VI. Due to lack of space, many proofs are omitted from this extended abstract, and can be found in the full version of the paper in arXiv:1208.1272v1.

II. Preliminaries

Notation. Suppose we are given an EDPwC instance $(G, \mathcal{M})$. We denote by $\mathcal{T}$ the set of vertices that participate in pairs in $\mathcal{M}$, and we call them *terminals*. Let OPT denote the maximum number of demand pairs that can be simultaneously routed with congestion at most 2. Given any subset $\mathcal{M}' \subseteq \mathcal{M}$ of the demand pairs, we denote by $\mathcal{T}(\mathcal{M}') \subseteq \mathcal{T}$ the subset of terminals participating in the pairs in $\mathcal{M}'$.

For any subset $S \subseteq V$ of vertices, we denote by $\mathrm{out}_G(S) = E_G(S, V \setminus S)$, and by $E_G(S)$ the subset of edges with both endpoints in S, omitting the subscript G when clear from context. Throughout the paper, we say that a random event succeeds w.h.p., if the probability of its success is $(1 - 1/\mathrm{poly}(n))$. All logarithms are to the base of 2.

Let $\mathcal{P}$ be any collection of paths in graph G. We say that paths in $\mathcal{P}$ cause congestion η in G, iff for every edge $e \in E$, at most η paths in $\mathcal{P}$ contain e. Given a set $\mathcal{P}$ of paths connecting the edges of E_1 to the edges of E_2, we denote $\mathcal{P} : E_1 \rightsquigarrow_\eta E_2$ iff $\mathcal{P} = \{P_e \mid e \in E_1\}$, where e is the first edge on P_e, and the total congestion caused by the paths in $\mathcal{P}$ is at most η. If every edge of E_2 has at most one path terminating at it, then we denote $\mathcal{P} : E_1 \overset{1:1}{\rightsquigarrow}_\eta E_2$. We use a similar notation for path sets connecting subsets of vertices to each other, or a subset vertices with a subset of edges.

Given a subset S of vertices and two subsets $E_1, E_2 \subseteq \mathrm{out}(S)$ of edges, we say that a set $\mathcal{P} : E_1 \rightsquigarrow_\eta E_2$ of paths is contained in S iff all inner edges on every path in $\mathcal{P}$ belong to $G[S]$.

Reduction to Well-Linked Instances. Given an instance $(G, \mathcal{M})$, we can assume w.l.o.g. that every terminal $t \in \mathcal{T}$ participates in exactly one source-sink pair: otherwise, for

each demand pair in which t participates, we can add a new terminal connected to t to the graph, that will replace t in the demand pair. Similarly, we can assume that the degree of every terminal is exactly 1, and the degree of every non-terminal vertex is at most 4. In order to achieve the latter property, we replace every vertex v whose degree $d_v > 4$ with a $d_v \times d_v$ grid, and connect the edges incident on v to the vertices of the first row of the grid. It is easy to verify that these transformations do not affect the solution value.

The notion of well-linkedness has been widely used in graph decomposition and routing, see e.g. [14], [15], [11]. While the main idea is similar, the definition details differ from paper to paper. Our definition of well-linkedness is similar to that of [12].

Definition 1. Let S be any subset of vertices of a graph G. For any integer k_1, for any $0 < \alpha \leq 1$, we say that set S is (k_1, α)-well-linked iff for any pair $T_1, T_2 \subseteq \text{out}(S)$ of disjoint subsets of edges, with $|T_1| + |T_2| \leq k_1$, for any partition (X, Y) of S with $T_1 \subseteq \text{out}(X)$ and $T_2 \subseteq \text{out}(Y)$, $|E_G(X, Y)| \geq \alpha \cdot \min\{|T_1|, |T_2|\}$.

Suppose a set S is not (k_1, α)-well-linked. We say that a partition (X, Y) of S is a (k_1, α)-*violating partition*, iff there are two subsets $T_1 \subseteq \text{out}(X) \cap \text{out}(S)$, $T_2 \subseteq \text{out}(Y) \cap \text{out}(S)$ of edges with $|T_1| + |T_2| \leq k_1$, and $|E_G(X, Y)| < \alpha \cdot \min\{|T_1|, |T_2|\}$.

Definition 2. Given a graph G, a subset S of its vertices, a parameter $\alpha > 0$, and a subset $\Gamma \subseteq \text{out}(S)$ of edges, we say that S is α-well-linked for Γ, iff for any partition (A, B) of S, $|E(A, B)| \geq \alpha \cdot \min\{|\Gamma \cap \text{out}(A)|, |\Gamma \cap \text{out}(B)|\}$. We say that the set S is α-well-linked iff it is α-well-linked for the set $\text{out}(S)$ of edges.

Notice that if $|\text{out}(S)| \leq k_1$, then S is α-well-linked iff it is (k_1, α)-well-linked.

Similarly, if we are given a graph G and a subset $\mathcal{T}$ of its vertices called terminals, we say that G is α-well linked for $\mathcal{T}$, iff for any partition (A, B) of $V(G)$, $|E(A, B)| \geq \alpha \cdot \min\{|A \cap \mathcal{T}|, |B \cap \mathcal{T}|\}$. Notice that if G is α-well-linked for $\mathcal{T}$, then for any pair $(\mathcal{T}_1, \mathcal{T}_2)$ of subsets of $\mathcal{T}$ with $|\mathcal{T}_1| = |\mathcal{T}_2|$, we can efficiently find a collection $\mathcal{P}$ of paths, $\mathcal{P} : \mathcal{T}_1 \overset{1:1}{\rightsquigarrow}_{\lceil 1/\alpha \rceil} \mathcal{T}_2$. This follows from the min-cut max-flow theorem and the integrality of flow.

Chekuri, Khanna and Shepherd [16], [14] have shown an efficient algorithm, that, given any EDP instance $(G, \mathcal{M})$, partitions it into a number of sub-instances $(G_1, \mathcal{M}_1), \ldots, (G_\ell, \mathcal{M}_\ell)$, such that, on the one hand, each instance G_i is 1-well-linked for the set of terminals participating in $\mathcal{M}_i$, and on the other hand, the sum of the values of the optimal fractional solutions in all these instances is $\Omega(\mathsf{OPT}/\log^2 k)$. Therefore, it is enough to find a polylogarithmic approximation with congestion 2 in each such sub-instance separately. We say an instance $(G, \mathcal{M})$ is a *well-linked instance* if, every vertex in G has degree at most 4, every terminal has degree exact 1 and participate in exactly one pair in $\mathcal{M}$, and G is 1-well-linked for the set $\mathcal{T}$ of terminals. From the above arguments, we only need to consider well-linked instances.

Sparsest Cut and the Flow-Cut Gap. Suppose we are given a graph $G = (V, E)$, and a subset $\mathcal{T} \subseteq V$ of k terminals. The sparsity of a cut $(S, \overline{S})$ in G is $\Phi(S) = \frac{|E(S,\overline{S})|}{\min\{|S \cap \mathcal{T}|, |\overline{S} \cap \mathcal{T}|\}}$, and the value of the sparsest cut in G is defined to be: $\Phi(G) = \min_{S \subset V}\{\Phi(S)\}$. The goal of the sparsest cut problem is, given an input graph G and a set $\mathcal{T}$ of terminals, to find a cut of minimum sparsity. Arora, Rao and Vazirani [17] have shown an $O(\sqrt{\log k})$-approximation algorithm for the sparsest cut problem. We denote this algorithm by $\mathcal{A}_{\text{ARV}}$, and its approximation factor by $\alpha_{\text{ARV}}(k) = O(\sqrt{\log k})$.

A problem dual to sparsest cut is the maximum concurrent flow problem. For the above definition of the sparsest cut problem, the corresponding variation of the concurrent flow problem asks to find the maximum value λ, such that every pair of terminals can send λ/k flow units to each other simultaneously with no congestion. The flow-cut gap is the maximum ratio, in any graph, between the value of the minimum sparsest cut and the maximum value λ of concurrent flow. The value of the flow-cut gap in undirected graphs, that we denote by $\beta(k)$ throughout the paper, is $\Theta(\log k)$ [18], [19], [20], [21]. Therefore, if $\Phi(G) = \alpha$, then every pair of terminals can send $\frac{\alpha}{k\beta(k)}$ flow units to each other with no congestion. Equivalently, every pair of terminals can send $1/k$ flow units to each other with congestion at most $\beta(k)/\alpha$. Moreover, any matching on the set $\mathcal{T}$ of terminals can be fractionally routed with congestion at most $2\beta(k)/\alpha$.

Given a graph G, a subset $S \subseteq V(G)$ of vertices, and a subset $\Gamma \subseteq \text{out}(S)$ of edges, we define an instance $\mathsf{SC}(G, S, \Gamma)$ of the sparsest cut problem as follows. First, we sub-divide every edge $e \in \Gamma$ by a new vertex t_e, and we let $\mathcal{T}(\Gamma) = \{t_e \mid e \in \Gamma\}$. Let $G_S(\Gamma)$ be the subgraph of the resulting graph, induced by $S \cup \mathcal{T}(\Gamma)$. The instance $\mathsf{SC}(G, S, \Gamma)$ of the sparsest cut problem is defined over the graph $G_S(\Gamma)$, where the vertices of $\mathcal{T}(\Gamma)$ serve as terminals. Observe that for all $\alpha \leq 1$, the value of the sparsest cut in $\mathsf{SC}(G, S, \Gamma)$ is at least α iff set S is α-well-linked with respect to Γ in graph G. If $\Gamma = \text{out}_G(S)$, then we will denote the corresponding instance of the sparsest cut problem by $\mathsf{SC}(G, S)$.

Expanders and the Cut-Matching Game. We say that a (multi)-graph $G = (V, E)$ is an α-expander, iff $\min_{\substack{S \subseteq V: \\ |S| \leq |V|/2}} \left\{ \frac{|E(S,\overline{S})|}{|S|} \right\} \geq \alpha$.

We use the cut-matching game of Khandekar, Rao and Vazirani [22]. In this game, we are given a set V of N vertices, where N is even, and two players: a cut player, whose goal is to construct an expander X on the set V of vertices, and a matching player, whose goal is to delay its construction. The game is played in iterations. We start with the graph X containing the set V of vertices, and no edges. In each iteration j, the cut player computes a bi-partition (A_j, B_j) of V into two equal-sized sets, and the matching player returns some perfect matching M_j between the two sets. The edges of M_j are then added to X. Khandekar, Rao and Vazirani have shown that there is a strategy for the cut player, guaranteeing that after $O(\log^2 N)$ iterations we obtain a $\frac{1}{2}$-expander w.h.p. Subsequently, Orecchia et al. [23] have shown the following improved bound:

Theorem 2 ([23]): There is a probabilistic algorithm for the cut player, such that, no matter how the matching player plays, after $\gamma_{\text{CMG}}(N) = O(\log^2 N)$ iterations, graph X is an $\alpha_{\text{CMG}}(N) = \Omega(\log N)$-expander, with constant probability.

Parameters. We now define some global parameters that will be used in our algorithm. Let $\gamma_{\text{CMG}} = \gamma_{\text{CMG}}(k) = O(\log^2 k)$ be the parameter for the number of iterations in the cut-matching game from Theorem 2. Let $\gamma = 2^{24}\gamma^4_{\text{CMG}}$. We will also use the following two parameters for well-linkedness: $\alpha = \frac{1}{2^{11}\gamma \log k} = \Omega\left(\frac{1}{\log^9 k}\right)$, used to perform the well-linked decomposition, and $\alpha_{\text{WL}} = \frac{\alpha}{\alpha_{\text{ARV}}(k)} = \Omega\left(\frac{1}{\log^{9.5} k}\right)$ - the well-linkedness factor we achieve. We use a parameter $k_1 = \frac{k}{192\gamma^3 \log \gamma} = \frac{k}{\text{poly}\log k}$, and we assume that the parameter k is large enough, so $k_1 > 4/\alpha$ (otherwise we can simply route one demand pair to obtain an $O(\text{poly}\log k)$-approximation). We say that a cluster $C \subseteq V(G)$ is *large* iff $|\text{out}(C)| \geq k_1$, and *small* otherwise.

III. Algorithm Overview

Chekuri, Khanna and Shepherd [16], [14], [24] have suggested the following high-level approach to solve EDP instances $(G, \mathcal{M})$, where G is well-linked for the terminals. They start by defining a graph called a *crossbar*: a graph H with a subset $Y \subseteq V(H)$ of vertices is called a crossbar with congestion c, iff any matching over the vertices of Y can be routed with congestion at most c in H. They then note that if we could show an algorithm that finds a crossbar (H, Y) in graph G, with $|Y| = k/\text{poly}\log k$, and constant congestion, then we can obtain a polylogarithmic approximation to EDPwC with constant congestion. An algorithm for constructing such a crossbar with a constant congestion follows from the recent work of [12].

We follow this approach, and define a structure that we call a *good crossbar*, which gives slightly stronger properties than the general crossbar defined above. The formal definition is as follows (see Figure 1).

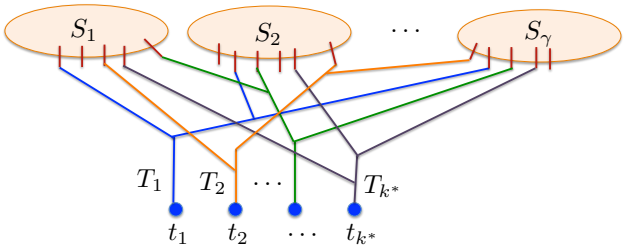

Figure 1. A good crossbar

Definition 3. Given a well-linked instance $(G, \mathcal{M})$ with $|\mathcal{M}| = k$ and a parameter $k^* = k/\text{poly}\log k$, a *good crossbar* consists of the following three components:

1) A family $\mathcal{S}^* = \{S^*_1, \ldots, S^*_{\gamma_{\text{CMG}}}\}$ of disjoint subsets of **non-terminal** vertices. Each set $S^*_j \in \mathcal{S}^*$ is associated with a subset $\Gamma^*_j \subseteq \text{out}(S^*_j)$ of $2k^*$ edges, and S^*_j is 1-well-linked for Γ^*_j.
2) A subset $\mathcal{M}^* \subseteq \mathcal{M}$ of k^* demand pairs. Let $\mathcal{T}^* = \mathcal{T}(\mathcal{M}^*)$ be the corresponding set of terminals.
3) A collection $\tau^* = \{T_1, \ldots, T_{2k^*}\}$ of trees in graph G. Each tree $T_i \in \tau^*$, contains a distinct terminal $t_i \in \mathcal{T}^*$, and for each $1 \leq j \leq \gamma_{\text{CMG}}$, tree T_i contains a distinct edge $e_{i,j} \in \Gamma^*_j$. In other words, $\mathcal{T}^* = \{t_1, \ldots, t_{2k^*}\}$, where $t_i \in T_i$ for each $1 \leq i \leq 2k^*$; and for each $1 \leq j \leq \gamma_{\text{CMG}}$, $\Gamma^*_j = \{e_{1,j}, \ldots, e_{2k^*,j}\}$, where $e_{i,j} \in T_i$ for each $1 \leq i \leq 2k^*$.

We say that the congestion of the good crossbar is c iff every edge of G participates in at most c trees in τ^*, while every edge in $\bigcup_{j=1}^{\gamma_{\text{CMG}}} E(S^*_j)$ belongs to at most $c-1$ such trees.

Chuzhoy [12] has implicitly defined a good crossbar, and has shown that, given a good crossbar that causes congestion c, there is an efficient randomized algorithm to route $\Omega(k^*/\text{poly}\log k)$ demand pairs with congestion at most c in graph G. This algorithm uses the cut-matching game of Khandekar, Rao and Vazirani [22] to embed an expander into G, and then finds a routing in this expander using the algorithm of Rao and Zhou [15]. She has also shown an efficient algorithm for constructing a good crossbar with congestion 14, thus obtaining an $O(\text{poly}\log k)$-approximation with congestion 14 for EDPwC.

We follow a similar approach here, except that we construct a good crossbar with congestion 2. Our main result is the following theorem:

Theorem 3: Assume we are given a well-linked instance $(G, \mathcal{M})$ with $|\mathcal{M}| = k$. Then there is an efficient randomized algorithm, that w.h.p outputs: (i) either a subset $\mathcal{M}' \subseteq \mathcal{M}$ of $k/\text{poly}\log k$ demand pairs and the routing of the pairs in $\mathcal{M}'$ with congestion at most 2 in G; or (ii) a good congestion-2 crossbar $(\mathcal{S}^*, \mathcal{M}^*, \tau^*)$.

Combining this with the result of [12], we obtain an $O(\mathrm{poly}\log k)$-approximation to EDPwC with congestion 2. From now on we focus on proving Theorem 3.

We now provide a high-level overview of the construction of the good crossbar of [12], and the barriers that need to be overcome to reduce the congestion to 2. The algorithm of [12] consists of three steps. In the first step, we construct a family $\mathcal{R} = \{S_1, \ldots, S_\gamma\}$ of disjoint clusters in $G \setminus \mathcal{T}$, where each cluster S_j is associated a set $\Gamma_j \subseteq \mathrm{out}(S_j)$ of $k^* = k/\mathrm{poly}\log k$ edges, for which S_j is 1-well-linked. Additionally, for every $1 \leq j \leq \gamma$, we are given a flow F_j, sending $k^*/2$ flow units from Γ_j to the terminals in $\mathcal{T}$. A family $\mathcal{R}$ of vertex subsets that has these properties is called a *good family of vertex subsets*. In the second step, we construct a family $\tau' = \{T'_1, \ldots, T'_{k^*}\}$ of trees, where every tree $T_i \in \tau'$ contains a distinct terminals $t_i \in \mathcal{T}$, and a distinct edge $e_{i,j} \in \Gamma_j$ for each $1 \leq j \leq \gamma$. However, the terminals in set $\mathcal{T}' = \{t_i \mid T_i \in \tau'\}$ do not necessarily form source-sink pairs in $\mathcal{M}$. In order to construct the trees in τ', consider the graph G' obtained from G by the following operations: for every $1 \leq j \leq \gamma$, add a super-node v_j; for every $e \in \Gamma_j$, connect v_j to the end-point of e that is not in S_j. Then the problem of finding the set τ' of trees is similar to the problem of packing Steiner trees in G', with Steiner nodes being $V(G') \setminus \{v_1, v_2, \cdots, v_\gamma\}$. The existence of the flows $\{F_j\}_{j=1}^{\gamma}$ can be viewed as evidence for the existence of the set τ' of trees in G', and we can use existing algorithms to find it. The well-linkedness of the sets $S_j \in \mathcal{R}$ is then exploited to simulate the super-nodes v_j, in order to construct a collection τ' of trees in the original graph. Finally, in the third step, we select a subset $\mathcal{M}^* \subseteq \mathcal{M}$ of $k^*/2$ demand pairs, and connect all terminals participating in the pairs in $\mathcal{M}^*$ to the terminals in set $\mathcal{T}'$. The union of these new paths with the trees in τ' gives the final collection τ of trees.

There are several factors contributing to the accumulation of congestion in this construction. We mention the main two barriers to reducing the congestion to 2 here.

The first problem is that we have used each cluster S_j twice in constructing the set τ' of trees: once for packing the Steiner-trees and once for simulating the super nodes v_j. It seems that this is unavoidable from the properties of $\mathcal{R}$: on the one hand, we have to use S_j to simulate v_j; on the other hand, we cannot remove the clusters S_j from G' since the flows $F_{j'}$ from $\Gamma_{j'}$ to $\mathcal{T}$, for $j' \neq j$, might use S_j.

The second problem is that step 2 and step 3 are executed separately, each contributing to the total congestion. It appears that one has to incur a congestion of at least 2 when constructing the set τ' of trees, using current techniques. If the terminals in set $\mathcal{T}'$ do not form demand pairs in $\mathcal{M}$, then we need to additionally select a subset $\mathcal{M}^* \subseteq \mathcal{M}$ of the demand pairs, and to route the terminals participating in $\mathcal{M}^*$ to the terminals of $\mathcal{T}'$, thus increasing the congestion beyond 2.

In order to find a good family $\mathcal{F}$ of vertex subsets, the algorithm of [12] performs a number of iterations. In each iteration we start with what is called a legal contracted graph G'. This graph is associated with a collection $\mathcal{C}$ of disjoint subsets of non-terminal vertices of G, such that each set $C \in \mathcal{C}$ is well-linked for $\mathrm{out}(C)$, and G' is obtained from G by contracting every cluster $C \in \mathcal{C}$ into a super-node. Additionally, we require that for each cluster $C \in \mathcal{C}$, $|\mathrm{out}(C)|$ is small. We call such a clustering $\mathcal{C}$ a *good clustering*. At the beginning of the algorithm, $\mathcal{C} = \emptyset$ and $G' = G$. In every iteration, given a legal contracted graph G', the algorithm either computes a good family $\mathcal{F}$ of vertex subsets, or produces a new legal contracted graph G'', containing strictly fewer vertices than G'. This guarantees that after n iterations, the algorithm produces a good family $\mathcal{F}$ of vertex subsets.

In order to overcome the two problems mentioned above and avoid accumulating the congestion, we combine all three steps of the algorithm together. We define a potential function φ over collections $\mathcal{C}$ of disjoint non-terminal vertex subsets, where $\varphi(\mathcal{C})$ roughly measures the number of edges in the graph obtained from G by contracting every cluster in $\mathcal{C}$ into a super-node. The potential function φ has additional useful properties, that allow us to perform a number of standard operations (such as the well-linked decomposition) on the clusters of $\mathcal{C}$, without increasing the potential value.

Our algorithm also consists of a number of iterations (that we call phases). In each such phase, we start with some legal contracted graph G' and a corresponding good clustering $\mathcal{C}'$. We then either construct a good crossbar, or produce a new good clustering $\mathcal{C}''$ with $\varphi(\mathcal{C}'') < \varphi(\mathcal{C}')$, together with the corresponding new legal contracted graph G''. Each phase is executed as follows. We start with some clustering $\mathcal{C}^*$ of the vertices of G, where $\varphi(\mathcal{C}^*) < \varphi(\mathcal{C}')$, but $\mathcal{C}^*$ is not necessarily a good clustering. We then perform a number of iterations. In each iteration, we select a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of disjoint subsets of non-terminal vertices, that we treat as a potential good family vertex subsets, and we try to find a subset $\mathcal{M}^* \subseteq \mathcal{M}$ of k^* demand pairs and a family τ of trees, to complete the construction of a good crossbar. If we do not succeed in constructing a good crossbar in the current iteration, then we use the family $\mathcal{F}$ of vertex subsets to refine the current clustering $\mathcal{C}^*$, such that the potential of the new clustering goes down by a significant amount. This ensures that after polynomially-many iterations, we will succeed in either constructing a good crossbar, or a good clustering $\mathcal{C}^*$ with $\varphi(\mathcal{C}^*) < \varphi(\mathcal{C}')$.

This combination of all three steps of the algorithm of [12] appears necessary to overcome the two barriers described above. For example, it is possible that the family $\mathcal{F}$ of vertex

subsets is a good family, but we are still unable to extend it to a good crossbar (for example because of the problem of the paths in $\mathcal{P}_j$ using the edges of $G[S_{j'}]$, as described above). Still, we will be able to make progress in such cases by refining the current clustering $\mathcal{C}^*$. Similarly, we construct the trees and connect the terminals participating in pairs in $\mathcal{M}^*$ to them simultaneously, to avoid accumulating congestion. Again, whenever we are unable to do so, we will be able to refine the current clustering $\mathcal{C}^*$.

In the following section we define several types of clusterings the algorithm uses, the notion of the legal contracted graphs, and several operations on a given clustering, that are used throughout the algorithm.

IV. Vertex Clusterings and Legal Contracted Graphs

In this section we define several types of vertex clusterings that the algorithm uses, the notion of the legal contracted graph, and several operations on a given clustering, that are used throughout the algorithm.

Definition 4. Given a partition $\mathcal{C}$ of the vertices of $V(G)$ into clusters, we say that $\mathcal{C}$ is an *acceptable clustering* of G iff:

- Every terminal $t \in \mathcal{T}$ is in a separate cluster, that is, $\{t\} \in \mathcal{C}$;
- Each small cluster $C \in \mathcal{C}$ is α_{WL}-well-linked; and
- Each large cluster $C \in \mathcal{C}$ is a connected component.

An acceptable clustering that contains no large clusters is called a *good clustering*.

Definition 5. Given a good clustering $\mathcal{C}$ of G, let $H_{\mathcal{C}}$ be the graph obtained from G by contracting every cluster $C \in \mathcal{C}$ into a super-node v_C (remove self-loops, but keep parallel edges). Then we say that $H_{\mathcal{C}}$ is the *legal contracted graph* of G associated with $\mathcal{C}$.

The following claim was proved in [12]; the proof is omitted here.

Claim 1: If G' is a legal contracted graph for G, then $G' \setminus \mathcal{T}$ contains at least $k/3$ edges.

Potential Function on Clusterings. Given **any** clustering $\mathcal{C}$ of the vertices of G, we define a potential $\varphi(\mathcal{C})$ for this clustering. For any integer h, we define $\varphi(h)$ as follows. For $h < k_1$, $\varphi(h) = 4\alpha \log h$. In order to define $\varphi(h)$ for $h \geq k_1$, we consider the sequence $\{n_0, n_1, \ldots\}$ of numbers, where $n_i = \left(\frac{3}{2}\right)^i k_1$. The potentials for these numbers are $\varphi(n_0) = \varphi(k_1) = 4\alpha \log k_1 + 4\alpha$, and for $i > 0$, $\varphi(n_i) = 4\frac{\alpha k_1}{n_i} + \varphi(n_{i-1})$. Notice that for all i, $\varphi(n_i) \leq 12\alpha + 4\alpha \log k_1 \leq 8\alpha \log k_1 \leq \frac{1}{2^8\gamma}$.

We now partition all integers $h > k_1$ into sets $S_1, S_2, \ldots$, where set S_i contains all integers h with $n_{i-1} \leq h < n_i$. For $h \in S_i$, we define $\varphi(h) = \varphi(n_{i-1})$. This finishes the definition of potentials of integers. Clearly, for all h, $\varphi(h) \leq \frac{1}{2^8\gamma}$.

Assume now that we are given some edge $e \in E$. If both endpoints of e belong to the same cluster of $\mathcal{C}$, then we set its potential $\varphi(e) = 0$. Otherwise, if $e = (u, v)$, and $u \in C$ with $|\operatorname{out}(C)| = h$, while $v \in C'$ with $|\operatorname{out}(C')| = h'$, then we set $\varphi(e) = 1 + \varphi(h) + \varphi(h')$. We think of $\varphi(h)$ as the contribution of u, and $\varphi(h')$ the contribution of v to $\varphi(e)$. Notice that $\varphi(e) \leq 1.1$. Finally, we set $\varphi(\mathcal{C}) = \sum_{e \in E} \varphi(e)$.

Well-linked decomposition of small clusters. Suppose we are given **any** partition $\mathcal{C}$ of $V(G)$. Our first step is to show that we can perform a well-linked decomposition of small clusters in $\mathcal{C}$, without increasing the potential. The proof of the following theorem is omitted due to lack of space.

Theorem 4: Let $\mathcal{C}$ be any partition of $V(G)$, and let $C \in \mathcal{C}$ be any small cluster, such that $G[C]$ is connected. Then there is an efficient algorithm that finds a partition $\mathcal{W}$ of C into small clusters, such that each cluster $R \in \mathcal{W}$ is α_{WL}-well-linked, and additionally, if $\mathcal{C}'$ is a partition obtained from $\mathcal{C}$ by removing C and adding the clusters of $\mathcal{W}$ to it, then $\varphi(\mathcal{C}') \leq \varphi(\mathcal{C})$.

We denote the procedure given by Theorem 4 by $\mathsf{DECOMPOSE}(C)$.

Given an acceptable clustering $\mathcal{C}$ of G, we define two operations on $\mathcal{C}$, each of which produces a new acceptable clustering of G, whose potential is at most $\varphi(\mathcal{C}) - 2$.

Operation 1: Partitioning a large cluster. Suppose we are given an acceptable clustering $\mathcal{C}$ of G, a large cluster $C \in \mathcal{C}$, and a (k_1, α)-violating partition (X, Y) of C. Then $\mathsf{PARTITION}(\mathcal{C}, C, X, Y)$ returns a new acceptable clustering $\mathcal{C}'$. In this operation, we first replace C with X and Y in $\mathcal{C}$. Additionally, if any of the clusters X and Y become small, we perform the operation $\mathsf{DECOMPOSE}$ on that cluster and update $\mathcal{C}$ with the resulting partitioning. Clearly, the final clustering $\mathcal{C}'$ is an acceptable clustering. The proof of the following claim is omitted due to lack of space.

Claim 2: Let $\mathcal{C}'$ be the clustering returned by the operation $\mathsf{PARTITION}(\mathcal{C}, C, X, Y)$. Then $\varphi(\mathcal{C}') \leq \varphi(\mathcal{C}) - 2$.

Operation 2: Separating a large cluster. Let $C \in \mathcal{C}$ be a large cluster in an acceptable clustering $\mathcal{C}$. Assume further that we are given a cut (A, B) in graph G, with $C \subseteq A$, $\mathcal{T} \subseteq B$, and $|E_G(A, B)| < k_1/2$. We perform the following operation, that we denote by $\mathsf{SEPARATE}(\mathcal{C}, C, A)$.

Consider some cluster $S \in \mathcal{C}$. If S is a small cluster, but $S \setminus A$ is a large cluster, then we modify A by removing all

vertices of S from it. Notice that in this case, the number of edges in $E(S)$ that originally contributed to the cut (A, B), $|E(S \cap A, S \cap B)| > |\operatorname{out}(S) \cap E(A)|$ must hold, so $|\operatorname{out}(A)|$ only goes down as a result of this modification. We assume from now on that if $S \in \mathcal{C}$ is a small cluster, then $S \setminus A$ is also a small cluster. We build a new partition $\mathcal{C}'$ of $V(G)$ as follows. First, we add every connected component of $G[A]$ to $\mathcal{C}$. Notice that all these clusters are small, as $|\operatorname{out}(A)| < k_1/2$. Next, for every cluster $S \in \mathcal{C}$, such that $S \setminus A \neq \emptyset$, we add every connected component of $G[S \setminus A]$ to $\mathcal{C}'$. Notice that every terminal $t \in \mathcal{T}$ is added as a separate cluster to $\mathcal{C}'$. In our final step, we replace every small cluster $C \in \mathcal{C}'$ with the clusters produced by $\mathsf{DECOMPOSE}(C)$. Let $\mathcal{C}''$ be the resulting acceptable clustering. Notice that if $S \in \mathcal{C}''$ is a large cluster, then there must be some **large** cluster S' in the original partition $\mathcal{C}$ with $S \subseteq S'$.

Claim 3: Let $\mathcal{C}''$ be the clustering returned by the operation $\mathsf{SEPARATE}(\mathcal{C}, C, A)$. Then $\varphi(\mathcal{C}'') \leq \varphi(\mathcal{C}) - 1$.

Proof: In order to prove the claim, it is enough to prove that $\varphi(\mathcal{C}') \leq \varphi(\mathcal{C}) - 1$, by Theorem 4. We can bound the changes in the potential as follows:

- Every edge in $\operatorname{out}(A)$ contributes at most 1.1 to the potential of $\mathcal{C}''$, and there are at most $\frac{k_1-1}{2}$ such edges. These are the only edges whose potential in $\mathcal{C}''$ may be higher than their potential in $\mathcal{C}$.
- Every edge in $\operatorname{out}(C)$ contributed at least 1 to the potential of $\mathcal{C}'$, and there are at least k_1 such edges, since C is a large cluster.

Therefore, the decrease in the potential is at least $k_1 - \frac{1.1(k_1-1)}{2} \geq 1$. ■

V. The Algorithm

In this section we prove Theorem 3, by providing an efficient randomized algorithm, that w.h.p. either computes a subset $\mathcal{M}' \subseteq \mathcal{M}$ of $k/\operatorname{poly}\log k$ demand pairs and their routing with congestion at most 2 in G, or finds a good congestion-2 crossbar in G.

We maintain, throughout the algorithm, a good clustering $\mathcal{C}$ of G. Initially, $\mathcal{C}$ is a partition of $V(G)$, where every vertex of G belongs to a distinct cluster, that is, $\mathcal{C} = \{\{v\} \mid v \in V(G)\}$. Clearly, this is a good clustering. We then perform a number of phases. Each phase is executed as follows.

For simplicity let, $G' = H_{\mathcal{C}}$ be the legal contracted graph of G associated with $\mathcal{C}$. Let m be the number of edges in $G' \setminus \mathcal{T}$. From Claim 1, $m \geq k/3$. As a first step of each phase, we randomly partition the vertices in $G' \setminus \mathcal{T}$ into γ subsets $X_1, \ldots, X_\gamma$, where each vertex $v \in V(G') \setminus \mathcal{T}$ selects an index $1 \leq j \leq \gamma$ independently uniformly at random, and is then added to X_j. We need the following claim, that appeared in [12]; we omit the proof here due to lack of space.

Claim 4: With probability at least $\frac{1}{2}$, for each $1 \leq j \leq \gamma$, $|\operatorname{out}_{G'}(X_j)| < \frac{10m}{\gamma}$, while $|E_{G'}(X_j)| \geq \frac{m}{2\gamma^2}$.

We repeat the randomized partitioning procedure until the conditions of Claim 4 hold (which can be checked efficiently). From Claim 4, we are guaranteed to obtain the desired partition after $\operatorname{poly}(n)$ iterations w.h.p. Assume now that we are given a partition $X_1, \ldots, X_\gamma$ of $V(G') \setminus \mathcal{T}$, for which the conditions of Claim 4 hold. Then for each $1 \leq j \leq \gamma$, $|E_{G'}(X_j)| > \frac{|\operatorname{out}_{G'}(X_j)|}{20\gamma}$. Let $X'_j \subseteq V(G) \setminus \mathcal{T}$ be the set obtained from X_j, after we un-contract each cluster, that is, for each super-node $v_C \in X_j$, we replace v_C with the vertices of C. Notice that $\left\{X'_j\right\}_{j=1}^{\gamma}$ is a partition of $V(G) \setminus \mathcal{T}$.

Recall that $\mathcal{C}$ is the current good clustering of the vertices of G, and every cluster $C \in \mathcal{C}$ is either contained in X'_j, or it is disjoint from it. For each $1 \leq j \leq \gamma$, we construct an acceptable clustering $\mathcal{C}_j$ of G as follows. Initially, $\mathcal{C}_j$ is just the clustering obtained from $\mathcal{C}$ by replacing the clusters contained in X'_j with the connected components of X'_j. If any of these components C' is a small cluster, then we replace it with the collection of clusters returned by procedure $\mathsf{DECOMPOSE}(C')$. Clearly, $\mathcal{C}_j$ is an acceptable clustering, with the following property:

P1) If $C \in \mathcal{C}_j$ is a large cluster, then $C \subseteq X'_j$.

We need the following claim that bounds the potential of the clustering $\mathcal{C}_j$.

Claim 5: For each $1 \leq j \leq \gamma$, $\varphi(\mathcal{C}_j) \leq \varphi(\mathcal{C}) - 1$.

Proof: Let $\mathcal{C}'_j$ be the partition of $V(G)$, obtained as follows: we add to $\mathcal{C}'_j$ all clusters $C \in \mathcal{C}$ with $C \cap X_j = \emptyset$, and we add all connected components of $G[X_j]$ to $\mathcal{C}'_j$ (that is, $\mathcal{C}'_j$ is obtained like $\mathcal{C}_j$, except that we do not perform well-linked decompositions of the small clusters). From Theorem 4, it is enough to prove that $\varphi(\mathcal{C}'_j) \leq \varphi(\mathcal{C}) - 1$. The changes of the potential from $\mathcal{C}$ to $\mathcal{C}'_j$ can be bounded as follows:

- The edges in $E_{G'}(X_j)$ contribute at least 1 to $\varphi(\mathcal{C})$ and contribute 0 to $\varphi(\mathcal{C}'_j)$.
- The potential of edges in $\operatorname{out}_G(X'_j)$ may increase. The increase is at most $\varphi(n) \leq \frac{1}{2^8\gamma}$ per edge. So the total increase is at most $\frac{|\operatorname{out}_{G'}(X_j)|}{2^8\gamma} \leq \frac{|E_{G'}(X_j)|}{4}$. These are the only edges whose potential may increase.

Overall, the decrease in the potential is at least $\frac{|E_{G'}(X_j)|}{2} \geq \frac{m}{4\gamma^2} \geq \frac{k}{12\gamma^2} \geq 1$. ■

If there is some $1 \leq j \leq \gamma$ such that $\mathcal{C}_j$ is a good partition, then we replace $\mathcal{C}$ with $\mathcal{C}_j$ and start a new phase. Otherwise, we select any large cluster $S_j \in \mathcal{C}_j$ for each $1 \leq j \leq$

γ. We then consider the resulting collection $S_1, \ldots, S_\gamma$ of large clusters, and try to exploit them to construct a good crossbar. Notice that for each $1 \leq j \leq \gamma$, $S_j \subseteq X'_j$, the sets $S_1, \ldots, S_\gamma$ are mutually disjoint and they do not contain terminals, due to Property (P1). Our algorithm then uses the following theorem, whose proof appears in the following section:

Theorem 5: Suppose we are given a family $\mathcal{R} = \{S_1, S_2, \cdots, S_\gamma\}$ of disjoint large clusters in $G \setminus \mathcal{T}$. Then there is an efficient randomized algorithm, that w.h.p. computes one of the following:

- Either a subset $\mathcal{M}' \subseteq \mathcal{M}$ of $k/\operatorname{poly}\log k$ demand pairs, and a routing of pairs in $\mathcal{M}'$ with congestion at most 2 in G;
- Or a good congestion-2 crossbar $(\mathcal{S}^*, \mathcal{M}^*, \tau^*)$;
- Or a (k_1, α)-violating partition (X, Y) of S_j, for some $1 \leq j \leq \gamma$;
- Or a cut (A, B) in G with $S_j \subseteq A$, $\mathcal{T} \subseteq B$ and $|E_G(A, B)| < k_1/2$, for some $1 \leq j \leq \gamma$.

We apply the algorithm in Theorem 5 to the current family $\{S_1, S_2, \cdots, S_\gamma\}$ of vertex subsets. If the algorithm returns either a routing of $\mathcal{M}'$, or a good congestion-2 crossbar, then we terminate the algorithm. Otherwise, we apply the appropriate action: $\mathsf{PARTITION}(\mathcal{C}_j, S_j, X, Y)$, or $\mathsf{SEPARATE}(\mathcal{C}_j, S_j, A)$ to obtain a new partition $\mathcal{C}'_j$ with $\varphi(\mathcal{C}'_j) \leq \varphi(\mathcal{C}_j) - 1$. Moreover, it is easy to see that this new clustering also has Property (P1): if the PARTITION operation is performed, then we only partition existing clusters; if the SEPARATE operation is performed, then the only large clusters in the new partition $\mathcal{C}'_j$ are subsets of large clusters in $\mathcal{C}_j$.

If all clusters in $\mathcal{C}'_j$ are small, then we replace $\mathcal{C}$ with $\mathcal{C}'_j$ and start a new phase. Otherwise, we replace $\mathcal{C}_j$ with $\mathcal{C}'_j$, let S_j be any large subset in the new partition $\mathcal{C}_j$, and apply Theorem 5 to the new collection $\{S_1, \ldots, S_\gamma\}$ of clusters, which are again guaranteed to be disjoint due to Property (P1). Notice that every time we apply Theorem 5, we reduce the potential of some clustering $\mathcal{C}_j$ by at least 1. Therefore, after applying the theorem polynomially many times, we are guaranteed to terminate the current phase of the algorithm, obtaining either a good clustering $\mathcal{C}'$ with $\varphi(\mathcal{C}') \leq \varphi(\mathcal{C}) - 1$, or a good congestion-2 crossbar, or a routing of a subset $\mathcal{M}'$ of $\Omega(k/\operatorname{poly}\log k)$ demand pairs with congestion at most 2.

In each phase, the potential of the clustering $\mathcal{C}$ is decreased by at least 1. Therefore, the algorithm terminates after a polynomial number of phases. In order to complete the proof of Theorem 3, it is now enough to prove Theorem 5.

VI. Proof of Theorem 5

Throughout the algorithm, we will sometimes be interested in routing flow across the sets $S_j \in \mathcal{R}$. Specifically, given two subsets $\Gamma, \Gamma' \subseteq \operatorname{out}(S_j)$ of edges, with $|\Gamma| = |\Gamma'| \leq k_1/2$, we will be interested in routing integrally the edges of Γ to the edges of Γ' inside S_j, with congestion at most $1/\alpha$. In other words, we will be looking for a set $\mathcal{P} : \Gamma \overset{1:1}{\rightsquigarrow}_{\lceil 1/\alpha \rceil} \Gamma'$ of paths contained in S_j. Notice that if such a set does not exist, then we can find a (k_1, α)-violating partition (X, Y) of S_j, by using the min-cut max-flow theorem. We can then return this partition and terminate the algorithm. Therefore, in order to simplify the exposition of the algorithm, we will assume that whenever the algorithm attempts to find such a set $\mathcal{P}$ of paths, it always succeeds.

We start by verifying that for each $1 \leq j \leq \gamma$, the vertices of S_j can send $k_1/2$ flow units with no congestion to the terminals. If this is not the case for some set S_j, then there is a cut (A, B) with $S_j \subseteq A$, $\mathcal{T} \subseteq B$ and $|E_G(A, B)| < k_1/2$. We then return the partition (A, B) of G and finish the algorithm. From now on we assume that each set S_j can send $k_1/2$ flow units with no congestion to the terminals.

The rest of the proof consists of three steps. In the first step, we construct a degree-3 tree $\tilde{T}$, whose vertex set is $V(\tilde{T}) = \{v_S \mid S \in \mathcal{R}'\}$, for a large enough family $\mathcal{R}' \subseteq \mathcal{R}$ of vertex subsets, and each edge $e = (v_S, v_{S'})$ in tree $\tilde{T}$ corresponds to a collection $\mathcal{P}_e$ of paths in G, connecting the vertices of S to the vertices of S'. Moreover, we will ensure that the paths in $\bigcup_{e \in E(\tilde{T})} \mathcal{P}_e$ only cause congestion 2 in G. In the second step, we find a subset $\mathcal{M}^* \subseteq \mathcal{M}$ of the demand pairs, and route the terminals in $\mathcal{T}(\mathcal{M}^*)$ to the vertices of $S \cup S'$, where (S, S') is some pair of vertex subsets in $\mathcal{R}'$. In the final third step, we construct a good congestion-2 crossbar. We only provide a sketch of the three steps, omitting the proofs of Theorem 6, 7 and 8. The complete proof appears in the full version of the paper.

A. Step 1: Constructing the Tree $\tilde{T}$

This step is summarized in the following theorem.

Theorem 6: There is an efficient algorithm, that either computes a (k_1, α)-violating partition of some set $S_j \in \mathcal{R}$, or finds a subset $\mathcal{R}' \subseteq \mathcal{R}$ of size $r = 8\gamma_{\text{CMG}}$, a tree $\tilde{T}$ of maximum degree 3 with vertex set $V(\tilde{T}) = \{v_S \mid S \in \mathcal{R}'\}$, and a collection $\mathcal{P}_e$ of $k_2 = \Omega\left(\frac{k_1 \alpha \cdot \alpha_{\text{WL}}}{\gamma^{3.5}}\right)$ paths in G for every edge $e \in E(\tilde{T})$, such that:

- For each edge $e = (v_S, v_{S'}) \in E(\tilde{T})$, every path $P \in \mathcal{P}_e$ connects a vertex of S to a vertex of S', and does not contain the vertices of $\bigcup_{S_j \in \mathcal{R}'} S_j$ as inner vertices.
- The paths in $\bigcup_{e \in E(\tilde{T})} \mathcal{P}_e$ cause congestion 2 in G.

B. Step 2: connecting the terminals

To simplify the notation, assume that $\mathcal{R}' = \{S_1, \ldots, S_r\}$. In this step we connect a subset of terminals to two subsets $S, S' \in \{S_1, \ldots, S_r\}$, using the following theorem.

Theorem 7: There is an efficient algorithm, that either finds a routing of a subset $\mathcal{M}' \subseteq \mathcal{M}$ of $k/\operatorname{poly}\log k$ demand pairs via edge-disjoint paths in G, or finds the following:

- A subset $\mathcal{M}_1 \subseteq \mathcal{M}$ of $k_4 = \Omega(k_2/r^2)$ demand pairs and a partition of $\mathcal{T}(\mathcal{M}_1)$ into two subsets $\mathcal{T}_1', \mathcal{T}_1''$ of size k_4 such that for each pair $(s,t) \in \mathcal{M}_1$, $s \in \mathcal{T}_1', t \in \mathcal{T}_1''$ or vice versa,
- A sub-tree T^* of $\tilde{T}$ of size $r' \geq 2\gamma_{\text{CMG}}$ with $V(T^*) = \{v_{S''} : S'' \in \mathcal{R}''\}$ for some $\mathcal{R}'' \subseteq \mathcal{R}'$, a vertex $v_{S'}$ of degree at most 2, and a vertex v_S of degree 1 in T^* (possibly $v_S = v_{S'}$),
- Two sets $\mathcal{P}_1' : \mathcal{T}_1' \overset{1:1}{\rightsquigarrow} \operatorname{out}(S)$, $\mathcal{P}_1'' : \mathcal{T}_1'' \overset{1:1}{\rightsquigarrow} \operatorname{out}(S')$ of paths in G, and
- For each edge $e \in E(T^*) \subseteq E(\tilde{T})$, a subset $\mathcal{P}_e' \subseteq \mathcal{P}_e$ of $\lfloor k_2/2 \rfloor$ paths.

Moreover, paths in $\mathcal{P}_1' \cup \mathcal{P}_1'' \cup \left(\bigcup_{e \in E(T^*)} \mathcal{P}_e'\right)$ do not contain any vertices of $\bigcup_{v_{\tilde{S}} \in V(T^*)} \tilde{S}$ as inner vertices, and they cause congestion 2 in G. Additionally, every edge in $\operatorname{out}(S) \cup \operatorname{out}(S')$ is used by at most one path in the set.

C. Step 3: Building the Good Crossbar

Assume w.l.o.g $\mathcal{R}'' = \{S_1, \ldots, S_{r'}\}$, where $r' \geq 2\gamma_{\text{CMG}}$. Consider some set $S_j \in \mathcal{R}''$, and let $\mathcal{P}$ be any collection of paths in G. We denote by $\Gamma_j(\mathcal{P}) \subseteq \operatorname{out}(S_j)$ the multi-set of edges, that appear as the first or the last edge on any path in $\mathcal{P}$. We use the following theorem:

Theorem 8: There is an efficient algorithm, that either computes a subset $\mathcal{M}' \subseteq \mathcal{M}_1$ of $k/\operatorname{poly}\log k$ demand pairs and their routing with congestion at most 2 in G, or a (k_1, α)-violating partition of some set $S_j \in \mathcal{R}''$, or it computes, for each edge $e \in E(T^*)$, a subset $\mathcal{P}_e'' \subseteq \mathcal{P}_e'$ of $k_5 = \Omega(\alpha_{\text{WL}}^2 k_2)$ paths, and two subsets $\mathcal{P}_2' \subseteq \mathcal{P}_1', \mathcal{P}_2'' \subseteq \mathcal{P}_1'$, such that:

- Each set $S_j \in \mathcal{R}''$ is 1-well-linked for the set $\Gamma_j(\mathcal{P}')$ of edges, where $\mathcal{P}' = \mathcal{P}_2' \cup \mathcal{P}_2'' \cup \left(\bigcup_{e \in E(T^*)} \mathcal{P}_e''\right)$.
- There is a subset $\mathcal{M}_2 \subseteq \mathcal{M}_1$ of size $\Omega(\alpha_{\text{WL}}^2) \cdot |\mathcal{M}_1| = \Omega(\alpha_{\text{WL}}^2 \cdot k_4)$ demand pairs such that $\mathcal{P}_2' : \mathcal{T}_2' \overset{1:1}{\rightsquigarrow} \operatorname{out}(S)$ and $\mathcal{P}_2'' : \mathcal{T}_2'' \overset{1:1}{\rightsquigarrow} \operatorname{out}(S')$, where $\mathcal{T}_2' = \mathcal{T}_1' \cap \mathcal{T}(\mathcal{M}_2)$ and $\mathcal{T}_2'' = \mathcal{T}_1'' \cap \mathcal{T}(\mathcal{M}_2)$.

Let $k_6 = |\mathcal{M}_2| = \Omega(\alpha_{\text{WL}}^2) k_4$. Notice that $2k_6 < k_5$. For each edge $e \in E(T^*)$, while $|\mathcal{P}''(e)| > 2k_6$, we discard arbitrary paths from $\mathcal{P}''(e)$, until $|\mathcal{P}''(e)| = 2k_6$ holds. We now assume that $|\mathcal{P}''(e)| = 2k_6$ for all $e \in E(T^*)$, and recall that $|\mathcal{P}_2'|, |\mathcal{P}_2''| = k_6$.

We are now ready to define the good crossbar in graph G. Let $\mathcal{S}^*$ contain all sets S_j, where the degree of vertex v_j in tree T^* is either 1 or 2 (excluding the set S). Notice that at least half the vertices of T^* must have this property, and therefore, $|S^*| \geq \gamma_{\text{CMG}}$. If $|S^*| > \gamma_{\text{CMG}}$, then we discard vertex subsets from S^* arbitrarily, until $|S^*| = \gamma_{\text{CMG}}$ holds.

Instead of defining the set τ^* of $2k_6$ trees explicitly, we specify a set of paths in graph G, whose disjoint union will give the collection τ^* of trees. First, all the paths in $\mathcal{P}_2' \cup \mathcal{P}_2'' \cup \left(\bigcup_{e \in E(T^*)} \mathcal{P}_e''\right)$ are included to construct τ^*.

Consider some degree-2 vertex v_j of T^*. Let e, e' be the two edges incident to v_j in T^*, and $\Gamma, \Gamma' \subseteq \operatorname{out}(S_j)$ be the sets of edges lying on the paths $\mathcal{P}_e''$ and $\mathcal{P}_{e'}''$, respectively. Since S_j is 1-well-linked for $\Gamma \cup \Gamma'$, we can find a set $\mathcal{Q}_j : \Gamma \overset{1:1}{\rightsquigarrow}_1 \Gamma'$ of paths contained in S_j. Then, paths in $\mathcal{Q}_j$ are included to construct τ^*.

For a degree-3 vertex v_j of T^*, define e, e', e'' and $\Gamma, \Gamma', \Gamma'' \subseteq \operatorname{out}(S_j)$ in the same way. Then, we can find 2 sets $\mathcal{Q}_j : \Gamma \overset{1:1}{\rightsquigarrow}_1 \Gamma'$ and $\mathcal{Q}_j' : \Gamma \overset{1:1}{\rightsquigarrow}_1 \Gamma''$ of paths in S_j. The paths in $\mathcal{Q}_j$ and $\mathcal{Q}_j'$ are included to construct τ^*.

Notice that the included paths already form $2k_6$ trees. The remaining task is to connect the $2k_6$ terminals to the $2k_6$ trees.

We first consider the case $S = S'$. Let e be the unique edge incident on v_S in tree T^* (recall that $v_{S'}$ has degree 1 in T^*). Let $\Gamma \subseteq \operatorname{out}(S)$ be the subset of edges lying on the paths in $\mathcal{P}_e''$, and $\Gamma' \subseteq \operatorname{out}(S)$ be the subset of edges lying on the paths in $\mathcal{P}_2' \cup \mathcal{P}_2''$. We can then find a set $\mathcal{Q}(v) : \Gamma \overset{1:1}{\rightsquigarrow}_1 \Gamma'$ of paths in set S. The paths in $\mathcal{Q}(v)$ are included to construct τ^*.

For the case $S \neq S'$, let e be some edge incident to v_S and e' be the unique edge incident to $v_{S'}$ in tree T^*. Let $\Gamma_1 \subseteq \operatorname{out}(S)$ be the subset of edges lying on the paths in $\mathcal{P}_2'$ and $\Gamma_2 \subseteq \operatorname{out}(S)$ be the subset of edges lying on the paths in $\mathcal{P}_e''$. Let $\Gamma_1' \subseteq \operatorname{out}(S')$ be the subset of edges lying on the paths in $\mathcal{P}_2''$ and $\Gamma_2' \subseteq \operatorname{out}(S)$ be the subset of edges lying on the paths in $\mathcal{P}_{e'}''$. Notice that $|\Gamma_1| = |\Gamma_1'| = k_6$ and $|\Gamma_2| = |\Gamma_2'| = 2k_6$. We discard some edges in Γ_2 and Γ_2' so that $|\Gamma_2| = |\Gamma_2'| = k_6$. Since S is 1-well-linked for $\Gamma_1 \cup \Gamma_2$ and S' is 1-well-linked for $\Gamma_1' \cup \Gamma_2'$, we can find 2 sets of paths $\mathcal{Q} : \Gamma_1 \overset{1:1}{\rightsquigarrow}_1 \Gamma_2$ and $\mathcal{Q}' : \Gamma_1' \overset{1:1}{\rightsquigarrow}_1 \Gamma_2'$. $\mathcal{Q}$ and $\mathcal{Q}'$ are included to construct τ^*. We discard edges in Γ_2 and Γ_2' in such a way that each tree in τ' will contain exactly 1 terminal in $\mathcal{T}(\mathcal{M}_2)$.

We have included all the paths that constitute τ^*. It is easy to check that the paths form $2k_6$ trees, each containing a distinct terminal in $\mathcal{T}(\mathcal{M}_2)$. Moreover, the $2k_6$ trees cause congestion 2 in G; each edge inside any cluster $S \in \mathcal{S}^*$ is used at most once by τ^*. The final good crossbar is $(\mathcal{S}^*, \mathcal{M}^* = \mathcal{M}_2, \tau^*)$.

REFERENCES

[1] N. Robertson and P. D. Seymour, "Outline of a disjoint paths algorithm," in *Paths, Flows and VLSI-Layout*. Springer-Verlag, 1990.

[2] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[3] 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, no. 1, pp. 137–146, 2006.

[4] N. Garg, V. V. Vazirani, and M. Yannakakis, "Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover," in *ICALP*, ser. Lecture Notes in Computer Science, A. Lingas, R. G. Karlsson, and S. Carlsson, Eds., vol. 700. Springer, 1993, pp. 64–75.

[5] M. Andrews and L. Zhang, "Hardness of the undirected edge-disjoint paths problem," in *STOC*, H. N. Gabow and R. Fagin, Eds. ACM, 2005, pp. 276–283.

[6] M. Andrews, J. Chuzhoy, V. Guruswami, S. Khanna, K. Talwar, and L. Zhang, "Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs," *Combinatorica*, vol. 30, no. 5, pp. 485–520, 2010.

[7] P. Raghavan and C. D. Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, pp. 365–374, December 1987. [Online]. Available: http://portal.acm.org/citation.cfm?id=45291.45296

[8] Y. Azar and O. Regev, "Strongly polynomial algorithms for the unsplittable flow problem," in *In Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2001, pp. 15–29.

[9] A. Baveja and A. Srinivasan, "Approximation algorithms for disjoint paths and related routing and packing problems," *Mathematics of Operations Research*, vol. 25, p. 2000, 2000.

[10] S. G. Kolliopoulos and C. Stein, "Approximating disjoint-path problems using packing integer programs," *Mathematical Programming*, vol. 99, pp. 63–87, 2004.

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

[12] J. Chuzhoy, "Routing in undirected graphs with constant congestion," in *Proceedings of the 44th symposium on Theory of Computing*, 2012, pp. 855–874.

[13] K. Kawarabayashi and Y. Kobayashi, "Breaking $O(n^{1/2})$-approximation algorithms for the edge-disjoint paths problem with congestion two," in *STOC*, L. Fortnow and S. P. Vadhan, Eds. ACM, 2011, pp. 81–88.

[14] C. Chekuri, S. Khanna, and F. B. Shepherd, "Multicommodity flow, well-linked terminals, and routing problems," in *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2005, pp. 183–192.

[15] S. Rao and S. Zhou, "Edge disjoint paths in moderately connected graphs," *SIAM J. Comput.*, vol. 39, no. 5, pp. 1856–1887, 2010.

[16] C. Chekuri, S. Khanna, and F. B. Shepherd, "The all-or-nothing multicommodity flow problem," in *STOC*, L. Babai, Ed. ACM, 2004, pp. 156–165, a full version at http://www.math.mcgill.ca/~bshepherd/PS/all.pdf.

[17] S. Arora, S. Rao, and U. V. Vazirani, "Expander flows, geometric embeddings and graph partitioning," *J. ACM*, vol. 56, no. 2, 2009.

[18] F. T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms," *Journal of the ACM*, vol. 46, pp. 787–832, 1999.

[19] N. Garg, V. Vazirani, and M. Yannakakis, "Approximate max-flow min-(multi)-cut theorems and their applications," *SIAM Journal on Computing*, vol. 25, pp. 235–251, 1995.

[20] N. Linial, E. London, and Y. Rabinovich, "The geometry of graphs and some of its algorithmic applications," *Proceedings of 35th Annual IEEE Symposium on Foundations of Computer Science*, pp. 577–591, 1994.

[21] Y. Aumann and Y. Rabani, "An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm," *SIAM J. Comput.*, vol. 27, no. 1, pp. 291–301, 1998.

[22] R. Khandekar, S. Rao, and U. V. Vazirani, "Graph partitioning using single commodity flows," in *STOC*, J. M. Kleinberg, Ed. ACM, 2006, pp. 385–390.

[23] L. Orecchia, L. J. Schulman, U. V. Vazirani, and N. K. Vishnoi, "On partitioning graphs via single commodity flows," in *Proceedings of the 40th annual ACM symposium on Theory of computing*, ser. STOC '08. New York, NY, USA: ACM, 2008, pp. 461–470. [Online]. Available: http://doi.acm.org/10.1145/1374376.1374442

[24] C. Chekuri, S. Khanna, and F. B. Shepherd, "Edge-disjoint paths in planar graphs," in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 71–80.