SIAM J. COMPUT.
Vol. 45, No. 4, pp. 1490–1532

ROUTING IN UNDIRECTED GRAPHS WITH CONSTANT CONGESTION*

JULIA CHUZHOY†

Abstract. Given an undirected graph $G = (V, E)$, a collection $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ of k pairs of its vertices, called demand pairs, and an integer c, the goal in the Edge-Disjoint Paths with Congestion problem is to connect the maximum possible number of demand pairs by paths, so that the maximum load on any edge (called edge congestion) does not exceed c. We show an efficient randomized algorithm to route $\Omega(\mathsf{OPT}/\mathrm{poly}\log k)$ demand pairs with congestion at most 14, where OPT is the maximum number of pairs that can be simultaneously routed on edge-disjoint paths. The algorithm in fact routes $\Omega(\mathsf{OPT}_{\mathsf{LP}}/\mathrm{poly}\log k)$ pairs, where $\mathsf{OPT}_{\mathsf{LP}}$ is the optimal value of the standard multicommodity flow relaxation for the problem. The best previous efficient algorithm that routed $\Omega(\mathsf{OPT}/\mathrm{poly}\log n)$ pairs required congestion $\mathrm{poly}(\log\log n)$, and for the setting where the maximum allowed congestion is bounded by a constant c, the best previous efficient algorithms could only guarantee the routing of $\Omega(\mathsf{OPT}/n^{1/c})$ pairs. We also introduce a new type of vertex sparsifier that we call an *integral flow sparsifier*; integral flow sparsifiers approximately preserve both fractional and integral routings. We show an algorithm that constructs such sparsifiers.

Key words. approximation algorithms, edge-disjoint paths, network routing

AMS subject classifications. 68R10, 05C85, 68W25

DOI. 10.1137/130910464

1. Introduction.

We study network routing problems in undirected graphs. In such problems, we are given an undirected n-vertex graph $G = (V, E)$ and a collection $\mathcal{M} = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$ of k pairs of vertices of G, called *source-sink pairs*, that we also refer to as *demand pairs*. In order to route a pair (s_i, t_i), we need to select a path connecting s_i to t_i in graph G. Given a routing of any subset of the demand pairs, its *congestion* is the maximum load on any edge, that is, the maximum number of paths containing the same edge. In general, we would like to route as many demand pairs as possible, while minimizing the edge congestion. These two conflicting objectives naturally give rise to several basic optimization problems.

One of the central routing problems is Edge-Disjoint Paths (EDP), where the goal is to route the maximum number of demand pairs on edge-disjoint paths (that is, with congestion 1). Robertson and Seymour [55, 56] have shown an efficient algorithm to solve this problem, when the number k of the demand pairs is bounded by a constant. However, when k is part of the input, it is NP-hard to even decide whether all pairs can be simultaneously routed on edge-disjoint paths [36, 27]. The best currently known approximation algorithm for the problem, due to Chekuri, Khanna, and Shepherd [18], achieves an $O(\sqrt{n})$-approximation factor, while the best current hardness of approximation is $\Omega(\log^{1/2-\epsilon} n)$ for any constant ϵ, unless $\mathsf{NP} \subseteq \mathsf{ZPTIME}(n^{\mathrm{poly}\log n})$ [3, 2]. We note that the standard multicommodity flow linear programming (LP) relaxation for EDP that is commonly used in approximation algorithms for network routing problems has an integrality gap of $\Omega(\sqrt{n})$ [31]. Interestingly, Rao and Zhou [54] have

*Received by the editors February 21, 2013; accepted for publication (in revised form) November 16, 2015; published electronically August 30, 2016. This work was supported in part by NSF CAREER grant CCF-0844872, a Sloan Research Fellowship, and NSF grant CCF-1318242. A preliminary version of this paper appeared in STOC 2012.

http://www.siam.org/journals/sicomp/45-4/91046.html

†Toyota Technological Institute, Chicago, IL 60637 (cjulia@ttic.edu).

shown an $O(\text{poly}\log n)$-approximation algorithm for EDP in graphs where the value of the global minimum cut is $\Omega(\log^5 n)$, by rounding the same LP relaxation.

In the EDP problem, we require that the congestion of the routing be the best possible (congestion 1), and the goal is to maximize the number of pairs routed subject to this constraint. On the other extreme is the Congestion Minimization problem, where we need to route all source-sink pairs while minimizing the edge congestion. The classical randomized rounding technique of Raghavan and Thompson [53] gives the best currently known approximation algorithm for this problem, whose approximation factor is $O(\log n/\log\log n)$. On the negative side, Andrews and Zhang [4] show that the problem is hard to approximate to within a factor of $\Omega(\frac{\log\log n}{\log\log\log n})$ unless $\mathsf{NP} \subseteq \mathsf{ZPTIME}(n^{\text{poly}\log n})$.

A problem that lies between these two extremes, and is a natural framework for studying the tradeoff between the number of pairs routed and the edge congestion, is the Edge-Disjoint Paths with Congestion problem (EDPwC). We say that an algorithm $\mathcal{A}$ achieves a factor-α approximation with congestion c for EDPwC if it routes at least OPT/α source-sink pairs, and the congestion of this routing is bounded by c, where OPT is the maximum number of the demand pairs that can be simultaneously routed on edge-disjoint paths. In particular, a very interesting question is whether, by slightly relaxing the capacity constraints, and allowing small edge congestion, we can significantly increase the number of pairs routed.

When the congestion c is allowed to be as high as $\Omega(\log n/\log\log n)$, the randomized rounding algorithm of Raghavan and Thompson [53] gives a constant-factor approximation for EDPwC. For smaller values of c, until recently, only $O(n^{1/c})$-approximation algorithms have been known [7, 8, 44]. In a recent breakthrough, Andrews [1] has shown an efficient randomized algorithm that routes $\Omega(\frac{\mathsf{OPT}}{\log^{61} n})$ pairs with congestion $O((\log\log n)^6)$. In another recent result, Kawarabayashi and Kobayashi [37] have shown an efficient algorithm that routes $\Omega(\frac{\mathsf{OPT}}{n^{3/7}})$ pairs with congestion 2, thus improving the best previously known $O(\sqrt{n})$-approximation for $c = 2$.

In this paper we show an efficient randomized algorithm that routes $\Omega(\frac{\mathsf{OPT}}{\log^{22.5} k \log\log k})$ demand pairs with congestion at most 14. We note that, on the negative side, Andrews et al. [2] have shown that for any fixed constant $\epsilon > 0$, for any $1 \le c \le O(\frac{\log\log n}{\log\log\log n})$, there is no $O((\log n)^{\frac{1-\epsilon}{c+1}})$-approximation algorithm for EDPwC with congestion c unless $\mathsf{NP} \subseteq \mathsf{ZPTIME}(n^{\text{poly}\log n})$. Therefore, the best approximation factor one may hope to achieve for EDPwC in the setting where the allowed congestion is bounded by a constant is polylogarithmic.

In a subsequent work, our main result was extended in [24] to an algorithm that achieves an $O(\text{poly}\log(k))$-approximation for EDPwC with congestion 2. Chekuri and Ene [15] considered a closely related, and more general, Node-Disjoint Paths problem, where the congestion is measured on vertices and not edges. They show an algorithm that achieves an $O(\text{poly}\log(k))$-approximation with constant congestion for this problem, building on some of the ideas from this paper. Some of the new technical ideas introduced in this paper have also led to new results in graph theory, including a new algorithm to decompose large-treewidth graphs into a number of disjoint large-treewidth subgraphs [12], a new proof of the Excluded Grid Theorem that achieves the first polynomial bounds on the size of the grid-minor [13], and a construction of degree-3 treewidth sparsifiers [14].

While our algorithm is guaranteed, with high probability,[1] to route at least $\Omega(\mathsf{OPT}/\operatorname{poly}\log k)$ of the demand pairs with constant congestion, we have no control over the choice of the pairs that are routed. In some applications it may be useful to be able to choose specific pairs for the algorithm to route beforehand. While we do not expect to be able to preselect an arbitrary collection of the demand pairs to be routed (from the NP-hardness of the decision version of EDP), under some conditions we can still have some control over their selection. We show that if a given set $\mathcal{T}$ of vertices is well-linked in the graph G (see a formal definition below), then we can efficiently find a partition $\mathcal{G}$ of $\mathcal{T}$ into groups of size $O(\operatorname{poly}\log k)$ each, with the following property: if we are given *any* collection $\mathcal{M} = \{(s_1, t_1), \ldots, (s_r, t_r)\}$ of demand pairs, with $\mathcal{M} \subseteq \mathcal{T} \times \mathcal{T}$, where for each group $U \in \mathcal{G}$, the vertices of U appear in at most one pair in $\mathcal{M}$, then there is an efficient randomized algorithm that with high probability routes *all* demand pairs in $\mathcal{M}$ with constant congestion.

We then turn to vertex flow sparsifiers. Given a graph G with a subset $\mathcal{T}$ of k vertices called terminals, and a set $D : \mathcal{T} \times \mathcal{T} \to \mathbb{R}^+$ of demands over pairs of vertices of $\mathcal{T}$, let $\eta(G, D)$ be the minimum congestion required to fractionally route the demands in D in graph G. We say that a graph H is a quality-q vertex flow sparsifier for $(G, \mathcal{T})$ iff $\mathcal{T} \subseteq V(H)$, and for any set D of demands over $\mathcal{T}$, $\eta(G, D) \leq \eta(H, D) \leq q\eta(G, D)$. Flow sparsifiers were first introduced by Moitra [49] and Leighton and Moitra [45]. One of their motivations was to obtain better and faster approximation algorithms for combinatorial optimization problems, whose solution value depends only on the congestion values $\eta(G, D)$ for various sets D of demands. The improvement is obtained by running the approximation algorithm on the sparsifier H instead of G, assuming that $|V(H)| \ll |V(G)|$. This results in a better running time and in better approximation factors for algorithms whose approximation factors depend on the input graph size. Several efficient algorithms are now known for constructing quality-$O(\log k/\log\log k)$ sparsifiers H with $V(H) = \mathcal{T}$ [11, 26, 48], and constant-quality sparsifiers when $V(H)$ may contain additional nonterminal vertices [22]. However, such sparsifiers do not preserve *integral routings*. For example, if we were to solve the EDP problem, or some other routing problem in graph H, we would not be guaranteed that we could transform this solution into an integral solution in graph G. This motivates our definition of *integral sparsifiers* that approximately preserve both fractional and integral routings. Suppose we are given any n-vertex graph $G = (V, E)$ with a subset $\mathcal{T}$ of k vertices called terminals. We say that a graph H is a quality-(q_1, q_2) integral flow sparsifier for G and $\mathcal{T}$ iff (1) $\mathcal{T} \subseteq V(H)$; (2) for any set D of demands over $\mathcal{T}$, $\eta(H, D) \leq q_1\eta(G, D)$ (so in particular if we scale the demands in D down by factor q_1, we can route them fractionally in H with no congestion); and (3) given any integral routing $\mathcal{P}$ of any set $\mathcal{M}$ of pairs of terminals in graph H with congestion c, there is an efficient randomized algorithm that with high probability finds an integral routing $\mathcal{P}'$ of $\mathcal{M}$ in G with congestion at most $q_2 \cdot c$. We show an efficient algorithm to construct integral sparsifiers H of quality (q_1, q_2) with $q_1 = O(\operatorname{poly}\log k)$, $q_2 = 31$, and $|V(H)| = O(d)$, where d is the sum of the degrees of all terminals.

Other related work. EDP and its variants have been studied extensively, and better approximation algorithms are known for several special cases. Some examples include planar graphs [28, 43, 42, 17, 19, 38, 57], trees [31, 21], and expander graphs [46, 10, 9, 41, 30].

We note that the approximability of routing problems is somewhat better under-

[1] Throughout the paper, we use "with high probability" to mean probability $1 - 1/\operatorname{poly}(n)$, where n is the number of the graph vertices.

stood in directed graphs. The EDP problem has $\tilde{O}\left(\min\left\{n^{2/3}, \sqrt{m}\right\}\right)$-approximation algorithms in directed graphs, where m is the number of the graph edges [16, 60, 40], and it is hard to approximate to within a factor of $\Omega\left(m^{1/2-\epsilon}\right)$ for any constant ϵ [32]. The randomized rounding technique of Raghavan and Thompson [53] gives an $O(\log n/\log\log n)$-approximation for directed Congestion Minimization, and the problem is hard to approximate to within a factor of $\Omega(\log n/\log\log n)$ unless $\mathsf{NP} \subseteq \mathsf{ZPTIME}(n^{\mathrm{poly}\log n})$ [5, 23]. As for EDPwC, the randomized rounding technique gives an $O(cn^{1/c})$-approximation [44, 59] for any congestion bound c. On the other hand, for any $1 \leq c \leq O(\frac{\log n}{\log\log n})$, there is no $n^{O(1/c)}$-approximation algorithm for the problem in directed graphs unless $\mathsf{NP} \subseteq \mathsf{ZPTIME}(n^{\mathrm{poly}\log n})$ [23].

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

THEOREM 1.1. *There is a randomized polynomial-time algorithm that, given an undirected graph G and a set $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of k pairs of vertices of G called demand pairs, with high probability finds a collection $\mathcal{P}$ of paths connecting $\Omega(\frac{\mathsf{OPT}}{\log^{22.5} k \log\log k})$ demand pairs with congestion at most 14, where OPT is the maximum number of the demand pairs that can be simultaneously routed on edge-disjoint paths in G.*

Our algorithm in fact routes $\Omega(\frac{\mathsf{OPT_{LP}}}{\log^{22.5} k \log\log k})$ demand pairs, where $\mathsf{OPT_{LP}}$ is the value of the optimal solution to the standard multicommodity flow LP relaxation for the problem, where each edge carries at most one flow unit. Since the integrality gap of this LP relaxation is $\Omega(\sqrt{n})$ for EDP when no congestion is allowed, our result shows that the integrality gap improves from polynomial to polylogarithmic if we allow a congestion of 14.

Remark. We compare the number of the demand pairs routed by our solution with congestion 14 to the number OPT of the demand pairs routed by the optimal solution with congestion 1. We have set up the definitions in this way in order to be consistent with prior work, and this gives a bicriteria approximation to EDPwC with congestion 14. However, since our algorithm is based on LP rounding, we obtain an $O\left(\log^{22.5} k \log\log k\right)$-approximation algorithm even when compared to the value OPT' of the optimal solution with congestion 14 (that is, we also obtain a real approximation). Indeed, if we denote by $\mathsf{OPT_{LP}}$ the value of the optimal solution of the multicommodity LP relaxation for our EDP instance, where we only allow congestion 1 on edges, then our algorithm returns a routing of $\Omega(\frac{\mathsf{OPT_{LP}}}{\log^{22.5} k \log\log k})$ demand pairs with congestion 14. Let $\mathsf{OPT'_{LP}}$ denote the value of the optimal solution to the multicommodity LP relaxation where we allow congestion 14. It is easy to see that $\mathsf{OPT'_{LP}} \leq 14\mathsf{OPT_{LP}}$, since, given a fractional routing with congestion 14, by scaling all flows down by factor 14, we obtain a fractional routing with congestion 1. Therefore, the number of the demand pairs routed by our algorithm is at least $\Omega(\frac{\mathsf{OPT'_{LP}}}{\log^{22.5} k \log\log k}) = \Omega(\frac{\mathsf{OPT}'}{\log^{22.5} k \log\log k})$, and we obtain a (real) $O(\mathrm{poly}\log k)$ approximation for EDPwC with congestion 14.

A basic notion used throughout the paper is that of well-linkedness. Well-linkedness and its variations have been used extensively in graph minor theory and in previous work on routing problems [52, 17, 54, 1]. We say that a set $\mathcal{T}$ of vertices is α-well-linked in a graph G iff for any partition (A, B) of $V(G)$, $|E(A, B)| \geq \alpha \cdot \min\{|\mathcal{T} \cap A|, |\mathcal{T} \cap B|\}$.

Suppose we are given a graph $G = (V, E)$, a set $\mathcal{T} \subseteq V$ of vertices called terminals, a partition $\mathcal{G}$ of $\mathcal{T}$, and a collection $\mathcal{M} = \{(s_1, t_1), \ldots, (s_r, t_r)\}$ of demand pairs, where

for each $1 \leq i \leq r$, $s_i, t_i \in \mathcal{T}$. We say that the demand set $\mathcal{M}$ is $(1, \mathcal{G})$-restricted iff for every group $U \in \mathcal{G}$, at most one demand pair (s_i, t_i) contains a terminal of U (and only one terminal of U may participate in this pair). Our next theorem allows us to preselect, to some extent, the demand pairs to be routed, if the set $\mathcal{T}$ of terminals is well-linked in G.

THEOREM 1.2. *Suppose we are given an n-vertex graph $G = (V, E)$, a subset $\mathcal{T} \subseteq V$ of k_0 vertices called terminals, such that $\mathcal{T}$ is α_0-well-linked in G for some $0 < \alpha_0 < 1$, and an integer $c \geq 1$. Then we can efficiently find a partition $\mathcal{G}$ of the terminals of $\mathcal{T}$ into groups of size $O(\frac{(\log k_0)^{21+11/c}}{\alpha_0})$, such that, given any set $\mathcal{M}$ of demand pairs over $\mathcal{T}$, where $\mathcal{M}$ is $(1, \mathcal{G})$-restricted, there is an efficient randomized algorithm that with high probability finds a routing of all pairs in $\mathcal{M}$ with congestion at most $14c + 1$.*

In particular, we can achieve congestion 15 with group size $O(\log^{32} k_0/\alpha_0)$, and if the group size is $O(\log^{22} k_0/\alpha_0)$, then the congestion is 155. Finally, the next theorem provides an algorithm for constructing integral sparsifiers.

THEOREM 1.3. *There is an efficient algorithm that constructs, for any graph G and set $\mathcal{T} \subseteq V(G)$ of k terminals, an integral sparsifier H of quality (q_1, q_2), with $q_1 = O(\text{poly} \log k)$, $q_2 = 31$, and $|V(H)| = O(d)$, where d is the sum of the degrees of all terminals.*

We now give an overview of our techniques and compare them to previous work. The starting point of the proof of Theorem 1.1 is the same as in the work of [17, 54, 1]. We start with the standard LP relaxation for the EDP problem on graph G, and we compute a partition of G into disjoint induced subgraphs $G_1, \ldots, G_r$. For each $1 \leq i \leq r$, we compute a subset $\mathcal{M}_i \subseteq \mathcal{M}$ of demand pairs that are contained in G_i, such that the corresponding set $\mathcal{T}_i$ of terminals, containing all vertices that participate in the pairs in $\mathcal{M}_i$, is $\frac{1}{2}$-well-linked in G_i, and, moreover, $\sum_{i=1}^{r} |\mathcal{M}_i| \geq \Omega(\frac{|\mathcal{M}|}{\log^2 k})$. An algorithm for efficiently computing such a decomposition was shown by Chekuri, Khanna, and Shepherd [17]. From now on, it is enough to find a good routing in each resulting subinstance G_i separately. To simplify notation, let G denote any such subinstance G_i, let $\mathcal{M}$ denote the set $\mathcal{M}_i$ of demand pairs, and let $\mathcal{T}$ denote the corresponding set $\mathcal{T}_i$ of terminals. Since set $\mathcal{T}$ is $\frac{1}{2}$-well-linked in G, graph G has good expansion properties with respect to $\mathcal{T}$. However, graph G may be far from being an expander, since it may contain many vertices besides the terminals. Intuitively, a natural approach is to embed an expander X, whose vertex set is $\mathcal{T}$, into the graph G. Each edge $e = (t_i, t_j)$ of the expander is mapped to a path P_e connecting t_i to t_j in G, and the congestion of the embedding is the maximum, over all edges $e' \in E(G)$, of the number of paths in $\{P_e \mid e \in E(X)\}$, containing e'. If we could find a low-congestion embedding of an expander X into G, then we could use existing algorithms for routing on expanders to find a low-congestion routing of a polylogarithmic fraction of the demand pairs in X, which in turn would give us a low-congestion routing of the same demand pairs in G. This general framework was first suggested by Chekuri, Khanna, and Shepherd [17], who proposed to embed a crossbar into the input graph G. Intuitively, a crossbar is a graph for which efficient algorithms to approximately compute integral routings are known. In particular, expander graphs can be viewed as crossbars. This framework was used by Rao and Zhou [54] and by Andrews [1]. A very useful tool in embedding an expander into any graph with a well-linked set $\mathcal{T}$ of terminals is the cut-matching game of Khandekar, Rao, and Vazirani [39]. In this game, we have two players: a cut player, who wants to construct an expander X, and a matching player, who tries to delay its construction. We start with X

containing only the set $V(X)$ of $2N$ vertices and no edges. In each iteration i, the cut player computes a partition (A_i, B_i) of $V(X)$ with $|A_i| = |B_i| = N$, and the matching player computes a matching M_i between A_i and B_i. The edges of M_i are then added to X. Khandekar, Rao, and Vazirani [39] have shown that no matter what the matching player does, there is a strategy for the cut player (that we denote by $\mathcal{A}_{\mathrm{CMG}}$), such that after $O(\log^2 N)$ iterations, X becomes an expander. Moreover, there is an efficient algorithm to compute the partitions (A_i, B_i), given the previous responses of the matching player. A natural approach to constructing an expander X and embedding it into the graph G using the cut-matching game is the following. We use the algorithm $\mathcal{A}_{\mathrm{CMG}}$ for the cut player, while the matching player is simulated by finding appropriate flows in G. Specifically, we let $V(X) = \mathcal{T}$. If (A_i, B_i) is the bipartition of $V(X)$ computed by the cut player, then we can send $|A_i| = |B_i|$ flow units from the terminals of A_i to the terminals of B_i with low congestion in graph G, using the fact that $\mathcal{T}$ is well-linked in G. We then use the resulting flow to define the matching M_i. This procedure can be used to both construct the expander X and embed it into G. In fact, Khandekar, Rao, and Vazirani use precisely this procedure in their algorithm for the sparsest cut problem.

One problem with this approach is that we need to compute $\Theta(\log^2 k)$ different flows in graph G, and together they may cause a polylogarithmic congestion. Moreover, the partitions that the cut player computes depend on the matchings computed in previous iterations, so we cannot attempt to route all these flows simultaneously in graph G with low congestion. Rao and Zhou [54] have proposed the following approach to overcome this difficulty. Let $\gamma = \Theta(\log^2 k)$ be the number iterations in the algorithm of [39]. We can build γ graphs $G_1, \ldots, G_\gamma$, where, for each $1 \leq i \leq \gamma$, $V(G_i) = V(G)$, and the sets $E(G_1), \ldots, E(G_\gamma)$ of edges form a partition of the edges in $E(G)$. If we can construct the family $G_1, \ldots, G_\gamma$ of graphs so that, in each graph G_i, the set of terminals is still well-linked, then we can now construct the expander X and embed it into G by using the cut-matching game of [39], where in each iteration i, matching M_i is computed by finding a flow from A_i to B_i in graph G_i. Since the edges of each set M_i are embedded into distinct graphs G_i, the congestion does not accumulate, and we obtain a good embedding of X into G. In order to construct the graphs G_i, Rao and Zhou use the random procedure of Karger [35], where each edge $e \in E$ is added to one of the graphs G_i uniformly at random. However, this procedure only works if the value of the global minimum cut in G is at least polylogarithmic. In order to overcome this difficulty, Andrews [1] used Raecke's tree decomposition technique [52]. Roughly speaking, he decomposes the graph G into a collection $\mathcal{C}$ of disjoint clusters, where each cluster $C \in \mathcal{C}$ has some useful properties that allow us to find good routings across the cluster C efficiently. Moreover, if H is the graph obtained from G by contracting each cluster $C \in \mathcal{C}$ into a single vertex, then the set of terminals is well-linked in H, and the global minimum cut in H is large, so we can use the algorithm of Rao and Zhou to complete the routing.

Our algorithm uses a slightly different way to embed an expander into G, somewhat similar to that in [54]. Specifically, each vertex $t \in V(X)$ is represented by a connected subgraph C_t of graph G that contains the terminal t. Each edge $e = (t, t') \in E(X)$ is represented by a path P_e connecting some vertex $v \in C_t$ to some vertex $v' \in C_{t'}$ in G. We ensure that each edge $e' \in E(G)$ only participates in a constant number of the subgraphs $\{C_t\}_{t \in V(X)}$ and paths $\{P_e\}_{e \in E(X)}$. Once we find such an embedding, we use *vertex-disjoint* routing in the expander X, which gives a low edge-congestion routing in the original graph G.

A major point of our departure from previous work is in how the expander X is constructed and embedded into G. A central notion in our algorithm is that of a family of good routers. Let $k' = \Theta(k/\operatorname{poly}\log k)$ be a parameter, where $k = |\mathcal{M}|$ is the number of the demand pairs. For any subset $S \subseteq V$ of vertices, let $\operatorname{out}(S)$ be the set of edges with exactly one endpoint in S. Given a subset $S \subseteq V$ of vertices of G, we say that S has the α-bandwidth property iff the edges of $\operatorname{out}(S)$ are α-well-linked in $G[S]$. (More formally, subdivide every edge $e \in \operatorname{out}(S)$ with a new vertex t_e, and consider the subgraph H_S of the resulting graph induced by $S \cup \mathcal{T}'$, where $\mathcal{T}' = \{t_e \mid e \in \operatorname{out}(S)\}$. We say that the set S has the α-bandwidth property iff $\mathcal{T}'$ is α-well-linked in H_S.) Similarly, if we are given a subset $\Gamma \subseteq \operatorname{out}(S)$ of edges, we say that S has the α-bandwidth property with respect to Γ iff the set $\{t_e \mid e \in \Gamma\}$ is α-well-linked in H_S.

We say that a subset $S \subseteq G$ of vertices is a *good router* iff there is a collection $\Gamma \subseteq \operatorname{out}(S)$ of k' edges, such that S has the α-bandwidth property with respect to Γ (where $\alpha = \Omega(1/\operatorname{poly}\log k)$), and moreover the edges in Γ can send $|\Gamma|$ flow units in G to the terminals in $\mathcal{T}$ with constant edge-congestion. A family $\mathcal{F}$ of vertex subsets is a *family of good routers* iff it contains γ mutually disjoint good routers $S_1, \ldots, S_\gamma$, where $\gamma = O(\log^2 k)$ is the parameter from the cut-matching game of [39].

Suppose we find a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers. For each $1 \le j \le \gamma$, let $\Gamma_j \subseteq \operatorname{out}(S_j)$ be the corresponding subset Γ of edges. In order to construct the expander X, we select a subset $\mathcal{T}' = \{t_1, \ldots, t_{k'}\} \subseteq \mathcal{T}$ of k' terminals, and we let $V(X) = \mathcal{T}'$. For each $1 \le i \le k'$, we then construct a connected subgraph C_i in graph G that contains, for each $1 \le j \le \gamma$, a distinct edge $e_{i,j} \in \Gamma_j$ and also contains the terminal t_i (see Figure 1). For each $1 \le j \le \gamma$, the edges $e_{1,j}, \ldots, e_{k',j}$ are all distinct, and we view the edge $e_{i,j}$ as the representative of terminal t_i for the set S_j. We also ensure that each edge of graph G only participates in a constant number of the subgraphs $\{C_i\}_{i=1}^{k'}$. For each i, C_i is viewed as representing the vertex t_i of X in graph G. In order to construct the expander X, we use the cut-matching game of [39], where in each iteration $1 \le j \le \gamma$, we use the subgraph $G[S_j]$ to route some matching M_j between the representatives of the terminals in A_j and B_j for the set S_j. This ensures that the congestion does not accumulate across different iterations. Finally, we show an efficient algorithm for finding a family $\mathcal{F}$ of good routers.

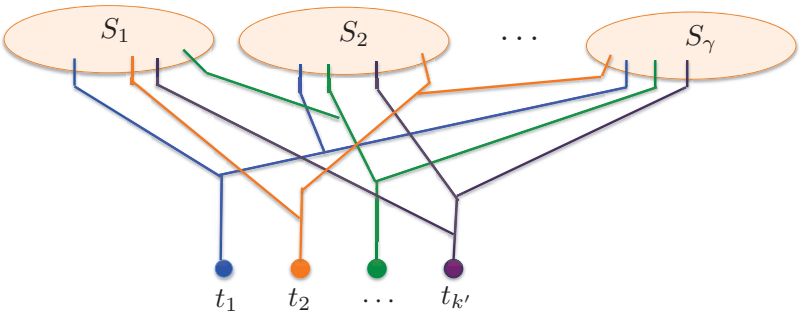


FIG. 1. *A family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers, together with the collection $\{C_1, \ldots, C_{k'}\}$ of subgraphs, where each subgraph C_i is a tree.*

Organization. Most of this paper is devoted to the proof of our main result, Theorem 1.1. We start with preliminaries in section 2 and provide the proof in

section 3. For convenience, a list of parameters is provided in Appendix D. The proofs of Theorems 1.2 and 1.3 appear in sections 4 and 5, respectively.

2. Preliminaries and notation. In this section we provide notation and basic results that we use to prove Theorem 1.1. We assume that we are given an undirected n-vertex graph $G = (V, E)$ and a set $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of k source-sink pairs, which we also refer to as demand pairs. 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 the demand pairs that can be simultaneously routed via edge-disjoint paths. Our goal is to connect $\Omega(\mathsf{OPT}/\operatorname{poly}\log k)$ distinct pairs in $\mathcal{M}$ with paths that cause congestion at most 14. Throughout the paper, all logarithms are to the base of 2. We sometimes refer to vertex subsets as *clusters*. We say that a cluster $C \subseteq V(G)$ is connected iff $G[C]$ is connected.

We perform a number of transformations to our graph to ensure that the maximum vertex degree in G is 4, the degree of every terminal is 1, and every terminal participates in one source-sink pair. We note that these are standard transformations that were used in prior work. We assume without loss of generality that every terminal of $\mathcal{T}$ participates in exactly one source-sink pair. Otherwise, if a terminal $v \in \mathcal{T}$ participates in $r > 1$ source-sink pairs, we can add r new terminals $t_1(v), \ldots, t_r(v)$, connect each of them to v with an edge, and use a distinct terminal in $\{t_1(v), \ldots, t_r(v)\}$ for each source-sink pair in which v participates. We also assume without loss of generality that the maximum vertex degree in G is 4 and that the degree of every terminal is 1. In order to achieve this, we perform the following simple transformation of graph G. If v is a terminal whose degree is greater than 1, then we add a new vertex u to graph G that connects to v with an edge and becomes a terminal instead of v. Next, we process the nonterminal vertices one by one. Let v be any such vertex, and assume that the degree of v is $d > 4$. Let $u_1, \ldots, u_d \in V$ be the neighbors of v. We replace v with a $d \times d$ grid Z_v, and we denote by $u'_1, \ldots, u'_d$ the vertices in the first row of Z_v. For each $1 \le i \le d$, we add an edge (u_i, u'_i). It is easy to verify that any solution to the EDP problem in the original graph can be transformed into a feasible routing of the same value and no congestion in the new graph, and any routing in the new graph with congestion η can be transformed into a routing in the original graph with the same congestion. Therefore, we assume from now on that the maximum vertex degree in G is 4, the degree of every terminal is 1, and every terminal participates in one source-sink pair.

For any subset $S \subseteq V$ of vertices, we denote by $\operatorname{out}_G(S) = E_G(S, V \setminus S)$ and by $E_G(S)$ the subset of edges with both endpoints in S. When clear from context, we omit the subscript G. Throughout the paper, we say that a random event succeeds with high probability if the probability of success is at least $(1 - 1/\operatorname{poly}(n))$.

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, and some edge belongs to η paths of $\mathcal{P}$. Assume that we are given a subset $S \subseteq V$ of vertices and a subset $E' \subseteq E$ of edges of G. We say that a collection $\mathcal{P}$ of paths connects the vertices of S to the edges of E' with congestion at most η, and we denote $\mathcal{P} : S \rightsquigarrow_\eta E'$ iff $\mathcal{P} = \{P_v \mid v \in S\}$, where path P_v has v as its first vertex and some edge of E' as its last edge, and $\mathcal{P}$ causes congestion at most η in G. In particular, each edge in E' serves as the last edge on at most η paths in $\mathcal{P}$. Similarly, given two subsets S, S' of vertices, if $\mathcal{P}$ is a collection of paths connecting every vertex of S to some vertex of S' with overall edge-congestion at most η, then we denote this by $\mathcal{P} : S \rightsquigarrow_\eta S'$. Finally, if $|S| = |S'| = |\mathcal{P}|$, and each path in $\mathcal{P}$ connects a distinct vertex of S to a distinct

vertex of S' with edge-congestion at most η, then we denote this by $\mathcal{P} : S \overset{1:1}{\rightsquigarrow}_{\eta} S'$. Similarly, we say that a flow F connects the vertices of S to the edges of E' with congestion at most η, and we denote $F : S \rightsquigarrow_{\eta} E'$ iff each vertex $v \in S$ sends one flow unit to the edges in E', and the flow F causes congestion at most η in G. Notice that each flow-path in F starts at a vertex of S and terminates at some edge $e \in E'$. We view edge e as part of the flow-path, so in particular each edge in E' receives at most η flow units. Notice that from the integrality of flow, for any integer $\eta \geq 1$, there is a flow $F : S \rightsquigarrow_{\eta} E'$ iff there is a collection $\mathcal{P} : S \rightsquigarrow_{\eta} E'$ of paths. We define flows and paths between subsets of edges similarly. Given two subsets S, S' of vertices, if $F : S \rightsquigarrow_{\eta} S'$ is a flow where every vertex of S sends one flow unit, and every vertex of S' receives one flow unit, then we denote this by $F : S \overset{1:1}{\rightsquigarrow}_{\eta} S'$.

We will often be interested in a scenario where we are given a subset S of vertices and two subsets $E_1, E_2 \subseteq \text{out}(S)$ of edges. We say that the flow $F : E_1 \rightsquigarrow_{\eta} E_2$ is *contained in* S iff every flow-path is completely contained in $G[S]$, except for its first and last edges, which belong to $\text{out}(S)$. Similarly, 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 of $\mathcal{P}$ belong to $G[S]$.

Given a graph $G = (V, E)$ and a set $\mathcal{T} \subseteq V$ of terminals, a set D of demands is a function $D : \mathcal{T} \times \mathcal{T} \to \mathbb{R}^+$, which specifies, for every unordered pair $t, t' \in \mathcal{T}$, a demand $D_{t,t'}$. We say that the set D of demands is γ-restricted iff for each $t \in \mathcal{T}$, the total demand $\sum_{t' \in \mathcal{T}} D_{t,t'} \leq \gamma$. Given a partition $\mathcal{G}$ of the set $\mathcal{T}$ of terminals, we say that the set D of demands is $(\gamma, \mathcal{G})$-restricted iff for each $U \in \mathcal{G}$ the total demand $\sum_{t \in U} \sum_{t' \in \mathcal{T}} D_{t,t'} \leq \gamma$. We say that the set D of demands is *integral* iff $D_{t,t'}$ is integral for each $t, t' \in \mathcal{T}$.

Given any set D of demands, a *fractional routing* of D is a flow F, where every unordered pair $t, t' \in \mathcal{T}$ sends $D_{t,t'}$ flow units to each other. Given an integral set D of demands, an *integral* routing of D is a collection $\mathcal{P}$ of paths, where for each unordered pair $(t, t') \in \mathcal{T}$ there are $D_{t,t'}$ paths connecting t to t' in $\mathcal{P}$. The congestion of this routing is the congestion caused by the set $\mathcal{P}$ of paths in G. Observe that any matching $\mathcal{M}$ on the set $\mathcal{T}$ of terminals defines a set D of demands where $D_{t,t'} = 1$ for $(t, t') \in \mathcal{M}$ and $D_{t,t'} = 0$ otherwise. We do not distinguish between the matching $\mathcal{M}$ and the set D of demands.

Sparsest cut and the flow-cut gap. Suppose we are given a graph $G = (V, E)$, with nonnegative weights w_v on vertices $v \in V$, and a subset $\mathcal{T} \subseteq V$ of k terminals, such that for all $v \notin \mathcal{T}$, $w_v = 0$. For any subset $S \subseteq V$ of vertices, let $w(S) = \sum_{v \in S} w(v)$. The sparsity of a cut $(S, \overline{S})$ in G is $\Phi(S) = \frac{|E(S,\overline{S})|}{w(S) \cdot w(\overline{S})}$, and the value of the sparsest cut in G is defined to be $\Phi(G) = \min_{S \subset V} \{\Phi(S)\}$. In the sparsest cut problem, the input is a graph G with nonnegative vertex weights, and the goal is to find a cut of minimum sparsity. Arora, Rao, and Vazirani [6] have shown an $O(\sqrt{\log k})$-approximation algorithm for the sparsest cut problem. We will often work with a special case of the sparsest cut problem where for each $t \in \mathcal{T}$, $w_t = 1$.

A problem dual to sparsest cut is the maximum concurrent flow problem. For the case where the weights of all terminals are unit, the goal in the maximum concurrent flow problem is to find the maximum value λ, such that every pair of terminals can send λ 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 concurrent flow. The value of the flow-cut gap in undirected graphs with k terminals, which we denote by $\beta(k)$ throughout the paper, is $\Theta(\log k)$ [46]. Therefore, if $\Phi(G) = \alpha$, then every pair of terminals can send $\alpha/\beta(k)$ flow units to each other with no congestion.

We will use a slightly different, but also standard, and roughly equivalent, definition of sparsity. Given any partition $(S,\overline{S})$ of V, the *sparsity* of the cut $(S,\overline{S})$ is $\Psi(S,\overline{S}) = \frac{|E(S,\overline{S})|}{\min\{w(S),w(\overline{S})\}}$. We then denote $\Psi(G) = \min_{S\subset V}\{\Psi(S,\overline{S})\}$. It is easy to see that if the weight of every terminal is 1, then $2\Psi(G)/k \geq \Phi(G) \geq \Psi(G)/k$. Therefore, if $\Psi(G)=\alpha$, then $\Phi(G)\geq\alpha/k$, and 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 $2\beta(k)/\alpha$. In the rest of the paper, we will use the latter definition of sparsity, and we will use the term cut sparsity and the value of sparsest cut to denote $\Psi(S,\overline{S})$ and $\Psi(G)$, respectively. The algorithm of [6] can still be used to obtain a cut of sparsity at most $O(\sqrt{\log k})\cdot\Psi(G)$ in G. We denote by $\mathcal{A}_{\text{ARV}}$ this algorithm and by $\alpha_{\text{ARV}}(k)=O(\sqrt{\log k})$ its approximation factor.

Routing on expanders. We say that a multigraph $G=(V,E)$ is an α-expander iff $\min_{\substack{S\subseteq V:\\ 0<|S|\leq|V|/2}}\left\{\frac{|E(S,\overline{S})|}{|S|}\right\}\geq\alpha$. There are many algorithms for routing on expanders, e.g., [46, 10, 9, 41, 30], which give different types of guarantees. For example, Frieze [30] has shown that if G is an r-regular graph (where r is a constant) with strong enough expansion properties, then there is an efficient randomized algorithm for routing *any* matching on any subset of $\Omega(n/\log n)$ of its vertices via edge-disjoint paths. We need a slightly different type of guarantee: the routing should be on *vertex-disjoint* paths, and the graph degree may be superconstant. Rao and Zhou [54] give such an algorithm, which is summarized in the next theorem. For completeness, we provide a proof sketch in Appendix A.

THEOREM 2.1 (Theorem 7.1 in [54]). *Let $G=(V,E)$ be any n-vertex d-regular α-expander. Assume further that n is even and that the vertices of G are partitioned into $n/2$ disjoint demand pairs $\mathcal{M}=\{(s_1,t_1),\ldots,(s_{n/2},t_{n/2})\}$. Then there is an efficient algorithm that routes $\Omega(\frac{\alpha^2 n}{\log n\cdot d^2})$ of the demand pairs on vertex-disjoint paths in G.*

The cut-matching game. We use the cut-matching game of Khandekar, Rao, and Vazirani [39]. 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 bipartition (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 with high probability. Subsequently, Orecchia et al. [51] showed the following improved bound.

THEOREM 2.2 (see [51]). *There is an efficient 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 $\Omega(\log N)$-expander, with constant probability.*

Well-linkedness, bandwidth property, and bandwidth decompositions. We define the the notion of well-linkedness, which has been used extensively in algorithms for network routing, e.g., in [52, 20, 17, 54, 1].

DEFINITION 2.3. *We say that a set $\mathcal{T}$ of vertices is α-well-linked in graph G iff for every partition (A,B) of $V(G)$, $|E_G(A,B)|\geq\alpha\cdot\min\{|A\cap\mathcal{T}|,|B\cap\mathcal{T}|\}$.*

The following is an easy observation about well-linkedness that follows from the

duality between cuts and flows.

OBSERVATION 2.4. *Set $\mathcal{T}$ of vertices is α-well-linked in graph G for any real number $0 < \alpha \leq 1$ iff for every pair $\mathcal{T}_1, \mathcal{T}_2$ of disjoint equal-sized subsets of vertices of $\mathcal{T}$, there is a flow $F : \mathcal{T}_1 \overset{1:1}{\rightsquigarrow}_{1/\alpha} \mathcal{T}_2$ in G.*

Next, we define *bandwidth property*—a notion closely related to well-linkedness and similar to the notion of bandwidth property defined in [52]. Intuitively, we say that a cluster S has the α-bandwidth property iff the boundary $\text{out}(S)$ of the cluster is α-well-linked in $G[S]$. The formal definition appears below.

DEFINITION 2.5. *Given a graph G, a subset S of its vertices, and a real number $0 < \alpha \leq 1$, we say that S has the α-bandwidth property iff for any partition (A, B) of S, if we denote by $T_A = \text{out}(A) \cap \text{out}(S)$ and by $T_B = \text{out}(B) \cap \text{out}(S)$, then $|E(A, B)| \geq \alpha \cdot \min\{|T_A|, |T_B|\}$.*

Given any subset $\Gamma \subseteq \text{out}(S)$ of edges, we say that S has the α-bandwidth property with respect to Γ iff for any partition (A, B) of S, $|E(A, B)| \geq \alpha \cdot \min\{|\text{out}(A) \cap \Gamma|, |\text{out}(B) \cap \Gamma|\}$.

We also need a more general notion of bandwidth property that we define below. Intuitively, it handles subsets S of vertices, where $|\text{out}(S)|$ may be large, but we only need to route small amounts of flow across S.

DEFINITION 2.6. *Let S be any subset of vertices of a graph G. For any integer $k > 0$ and for any real number $0 < \alpha < 1$, we say that set S has the (k, α)-bandwidth property iff for any pair $T_1, T_2 \subseteq \text{out}(S)$ of disjoint subsets of edges, with $|T_1| + |T_2| \leq k$, for any partition (X, Y) of S with $T_1 \subseteq \text{out}(X)$ and $T_2 \subseteq \text{out}(Y)$, $|E_G(X, Y)| \geq \alpha \cdot \min\{|T_1|, |T_2|\}$.*

Note that if $|\text{out}(S)| \leq k$, then set S has the (k, α)-bandwidth property iff it has the α-bandwidth property, and the two definitions of bandwidth property become equivalent.

OBSERVATION 2.7. *If $S \subseteq V(G)$ has the (k, α)-bandwidth property, then for any subset $T \subseteq \text{out}(S)$ of at most k edges, any matching on T can be fractionally routed inside S with congestion at most $2\beta(k)/\alpha$.*

Proof. We set up an instance of the sparsest cut problem on a graph H, obtained as follows: first, we subdivide every edge $e \in T$ by a vertex v_e, and we let $\Gamma = \{v_e \mid e \in T\}$ be the resulting set of new vertices. Let H be the subgraph of the resulting graph, induced by $S \cup \Gamma$. Consider an instance of the sparsest cut problem on graph H, with the set Γ of terminals. Since S has the (k, α)-bandwidth property, the value of the sparsest cut is at least α, and so any matching on T can be routed with congestion at most $2\beta(k)/\alpha$. □

Assume now that S does not have the (k, α)-bandwidth property. Then there must be a partition (X, Y) of S, such that $|E(X, Y)| < \alpha \cdot \min\{|\text{out}(X) \cap \text{out}(S)|, |\text{out}(Y) \cap \text{out}(S)|, k/2\}$. We say that (X, Y) is a (k, α)-*violating cut* for S.

The reason for defining the more general (k, α)-bandwidth property is the following. Given a set S of vertices of G, we will sometimes need to find a partition of S into a set $\mathcal{W}$ of clusters such that, on the one hand, each cluster $W \in \mathcal{W}$ has the α-bandwidth property, while on the other hand, $\sum_{W \in \mathcal{W}} |\text{out}(W)|$ is very close to $|\text{out}(S)|$. Procedures for finding such decompositions are well known (sometimes under the name of well-linked decomposition). The problem is that $|\text{out}(S)|$ may be much larger than k (for example, it can be close to n), and in order to obtain such a partition, we need to set $\alpha = 1/\operatorname{poly}\log n$. In our algorithm, since we are trying to obtain an $O(\operatorname{poly}\log k)$-approximation for EDPwC, including in scenarios where

$k \ll n$, this dependency of α on n is unacceptable, and we need $\alpha = 1/\operatorname{poly}\log k$. If we use the new (k,α)-bandwidth property instead of the standard α-bandwidth property, such a decomposition $\mathcal{W}$ exists, as we show below, with $\alpha = 1/\operatorname{poly}\log k$. The (k,α)-bandwidth property is sufficient for us, since we only use the bandwidth property of S in order to route small (up to k) amounts of flow through S.

We now discuss the bandwidth-decomposition procedure in more detail. Given a subset S of vertices of G, we would like to find a partition $\mathcal{W}$ of S such that every set in $W \in \mathcal{W}$ has the (k,α)-bandwidth property. We could do so using the standard well-linked decomposition procedures—for example, similar to those used in [52, 17]. However, in order to do so, we need to be able to check whether a given subset W of vertices has the (k,α)-bandwidth property, and if not, find a (k,α)-violating cut efficiently. We do not know how to do this, even approximately. Therefore, we will assume for now that we are given an oracle that finds a (k,α)-violating cut in a given subset of vertices, if such a cut exists. We describe the decomposition procedure and bound the number of edges $\sum_{W\in\mathcal{W}} |\operatorname{out}(W)|$ in the resulting decomposition. When we use this decomposition later in the algorithm, we will be interested in routing small amounts of flow (up to k) across the clusters of the decomposition. Whenever we will be unable to route this flow, we will naturally obtain a (k,α)-violating cut. Therefore, our algorithm itself will serve as an oracle to the decomposition procedure. We note that in the final decomposition $\mathcal{W}$, not all sets $W \in \mathcal{W}$ may have the (k,α)-bandwidth property, but we will be able to route the flow that we need to route across these clusters, and this is sufficient for us. We now describe the oracle-based decomposition procedure and analyze it.

We are given as input a subset S of vertices of G, an integer $k > 8$, and a real parameter $0 < \alpha < 1$. Throughout the decomposition procedure, we maintain a partition $\mathcal{W}$ of S, and at the beginning, $\mathcal{W} = \{S\}$. The algorithm proceeds as follows. As long as not all sets in $\mathcal{W}$ have the (k,α)-bandwidth property, our oracle computes a (k,α)-violating cut (X,Y) of one of the sets $W \in \mathcal{W}$. We then remove W from $\mathcal{W}$ and add X and Y to $\mathcal{W}$ instead. In the next theorem, we bound $\sum_{W\in\mathcal{W}} |\operatorname{out}(W)|$.

THEOREM 2.8. *Assume that $k > 8$, $\alpha < \frac{1}{48\log k}$, and let $\mathcal{W}$ be any partition of S produced over the course of the above algorithm. Then $\sum_{W\in\mathcal{W}} |\operatorname{out}(W)| \le |\operatorname{out}(S)|\,(1 + 32\alpha\log k)$.*

We emphasize that the bound on $\sum_{W\in\mathcal{W}} |\operatorname{out}(W)|$ holds for any partition produced over the course of the algorithm, and not just the final partition.

Proof. The proof uses standard arguments. Let $E^* = E(S) \cup \operatorname{out}(S)$. Throughout the algorithm, we assign nonnegative budgets $B(v,e)$ to all edges $e \in E^*$, and vertices $v \in S$, where v is an endpoint of e. Given a current partition $\mathcal{W}$ of S, the budgets are assigned as follows. For each set $W \in \mathcal{W}$, for each edge $e \in \operatorname{out}(W)$, and endpoint v of e lying in W, we set $B(v,e) = 8\alpha\log(|\operatorname{out}(W)|)$ if $|\operatorname{out}(W)| < k$, and we set $B(v,e) = 16\alpha\log k + 8\alpha(1 - \frac{k}{|\operatorname{out}(W)|})$ otherwise. All other budgets $B(v,e)$ are set to 0. It is easy to see that all budgets are nonnegative. It is now enough to prove that throughout the algorithm, the following invariant holds:

$$\sum_{e\in E^*} \sum_{v\in e\cap S} B(v,e) + \sum_{W\in\mathcal{W}} |\operatorname{out}(W)| \le |\operatorname{out}(S)|(1 + 32\alpha\log k). \tag{1}$$

Clearly, the budget value $B(v,e)$ for $e \in E^*$ and $v \in S \cap e$ never exceeds $24\alpha\log k$. At the beginning of the algorithm, the sum of all budgets $B(v,e)$ is therefore bounded by $24\alpha\log k|\operatorname{out}(S)|$, and so the invariant holds. We now consider some iteration, where cluster W was partitioned into clusters X and Y, and we assume without loss

of generality that $|\operatorname{out}(X)| \leq |\operatorname{out}(Y)|$. We assume that (1) holds at the beginning of the current iteration, and we prove that it continues to hold at the end of the iteration. Recall that from the definition of the (k,α)-violating cut, $|E(X,Y)| < \alpha \cdot \min\{|\operatorname{out}(W) \cap \operatorname{out}(X)|, k/2\}$. Notice also that $|\operatorname{out}(X)| \leq |\operatorname{out}(X) \cap \operatorname{out}(W)| + |E(X,Y)| < 1.1|\operatorname{out}(X) \cap \operatorname{out}(W)| \leq 0.55|\operatorname{out}(W)|$. We now consider three cases.

First, if $|\operatorname{out}(W)| < k$, then the total decrease in the budgets of pairs (v,e) where $e \in \operatorname{out}(X) \cap \operatorname{out}(W)$ and $v \in X \cap e$ is at least

$$\begin{aligned}
&|\operatorname{out}(X) \cap \operatorname{out}(W)| \cdot (8\alpha \log(|\operatorname{out}(W)|) - 8\alpha \log(|\operatorname{out}(X)|)) \\
&\quad = |\operatorname{out}(X) \cap \operatorname{out}(W)| \cdot 8\alpha \log(|\operatorname{out}(W)|/|\operatorname{out}(X)|) \\
&\quad \geq 6\alpha |\operatorname{out}(X) \cap \operatorname{out}(W)|,
\end{aligned}$$

since $|\operatorname{out}(X)| \leq 0.55|\operatorname{out}(W)|$. The total number of edges in $E(X,Y)$ is bounded by $\alpha|\operatorname{out}(X) \cap \operatorname{out}(W)|$, and each such edge contributes 2 to the summation $\sum_{W \in \mathcal{W}} |\operatorname{out}(W)|$. Additionally, for each edge $e \in E(X,Y)$, for each endpoint v of e, the budget $B(v,e)$ becomes at most $8\alpha \log k$. Therefore, the total increase in $\sum_{W \in \mathcal{W}} |\operatorname{out}(W)|$ and $\sum_{e \in E(X,Y)} \sum_{v \in e} B(v,e)$ is bounded by

$$|E(X,Y)|(2 + 16\alpha \log k) \leq 2.5|E(X,Y)| \leq 2.5\alpha |\operatorname{out}(X) \cap \operatorname{out}(W)|.$$

The budgets $B(v,e)$ where $e \in \operatorname{out}(W) \cap \operatorname{out}(Y)$ and $v \in e \cap Y$ can only decrease, and all other budgets remain unchanged. We conclude that the invariant (1) continues to hold at the end of the iteration.

The second case is when $|\operatorname{out}(W)| \geq k$, but $|\operatorname{out}(X)| < k$. In this case, for each edge $e \in \operatorname{out}(X) \cap \operatorname{out}(W)$, and endpoint $v \in X$ of e, budget $B(v,e)$ decreases by at least $8\alpha \log k$, and so the total decrease in budgets $\sum_{e \in \operatorname{out}(X) \cap \operatorname{out}(W)} \sum_{v \in X \cap e} B(v,e)$ is at least $8|\operatorname{out}(X) \cap \operatorname{out}(W)| \cdot \alpha \log k$. Similarly to the previous case, the total increase in $\sum_{W \in \mathcal{W}} |\operatorname{out}(W)|$ and $\sum_{e \in E(X,Y)} \sum_{v \in e} B(v,e)$ is bounded by

$$|E(X,Y)|(2 + 32\alpha \log k) \leq 3|E(X,Y)| \leq 3\alpha |\operatorname{out}(X) \cap \operatorname{out}(W)|.$$

Therefore, the invariant (1) continues to hold at the end of the iteration.

The third, and final, case is when $|\operatorname{out}(W)| \geq k$ and $|\operatorname{out}(X)| \geq k$. In this case, the decrease in $\sum_{e \in \operatorname{out}(X) \cap \operatorname{out}(W)} \sum_{v \in X \cap e} B(v,e)$ is at least

$$\begin{aligned}
&|\operatorname{out}(W) \cap \operatorname{out}(X)| \cdot \left(8\alpha \left(1 - \frac{k}{|\operatorname{out}(W)|}\right) - 8\alpha \left(1 - \frac{k}{|\operatorname{out}(X)|}\right)\right) \\
&\quad = |\operatorname{out}(W) \cap \operatorname{out}(X)| \cdot 8k\alpha \left(\frac{1}{|\operatorname{out}(X)|} - \frac{1}{|\operatorname{out}(W)|}\right) \\
&\quad \geq |\operatorname{out}(W) \cap \operatorname{out}(X)| \cdot \frac{3.6k\alpha}{|\operatorname{out}(X)|} \\
&\quad \geq 3.24k\alpha,
\end{aligned}$$

since $|\operatorname{out}(X)| \leq 0.55|\operatorname{out}(W)|$, and $|\operatorname{out}(W) \cap \operatorname{out}(X)| \geq 0.9|\operatorname{out}(X)|$ (as $|E(X,Y)| \leq 0.1|\operatorname{out}(W) \cap \operatorname{out}(X)|$). On the other hand, the total increase in $\sum_{W \in \mathcal{W}} |\operatorname{out}(W)|$ and $\sum_{e \in E(X,Y)} \sum_{v \in e} B(v,e)$ is bounded by

$$|E(X,Y)|(2 + 48\alpha \log k) \leq 3|E(X,Y)| \leq 3\alpha k.$$

The budgets $B(v,e)$ where $e \in \operatorname{out}(W) \cap \operatorname{out}(Y)$ and $v \in e \cap Y$ can only decrease, and all other budgets remain unchanged. Therefore, the invariant (1) continues to hold at the end of the iteration. ∎

Throughout the paper, we use $\alpha(k) = \frac{1}{2^{11} \cdot \gamma_{\text{CMG}}(k) \cdot \log k}$ and $\alpha_{\text{BW}}(k) = \alpha(k)/\alpha_{\text{ARV}}(k)$.

If $|\operatorname{out}(S)| \leq k$, then we can obtain a decomposition $\mathcal{W}$ of S, where each set $W \in \mathcal{W}$ has the $(k, \alpha_{\text{BW}}(k))$-bandwidth property efficiently, by using the algorithm $\mathcal{A}_{\text{ARV}}$ for sparsest cut as our oracle: In each iteration, for each $W \in \mathcal{W}$, we apply the algorithm $\mathcal{A}_{\text{ARV}}$ to the corresponding instance of the sparsest cut problem (where the edges of $\operatorname{out}(W)$ are viewed as terminals). If algorithm $\mathcal{A}_{\text{ARV}}$ returns a $(k, \alpha(k))$-violating cut (X, Y) for any set $W \in \mathcal{W}$, then we can proceed with the decomposition procedure as before. Otherwise, we are guaranteed that each set $W \in \mathcal{W}$ has the $\alpha_{\text{BW}}(k)$-bandwidth property. We therefore have the following corollary.

COROLLARY 2.9. *Let S be any subset of vertices of G such that $|\operatorname{out}(S)| \leq k$, where $k \geq 8$. Then we can efficiently find a partition $\mathcal{W}$ of S such that for each $W \in \mathcal{W}$, $|\operatorname{out}(W)| \leq k$, and W has the $\alpha_{\text{BW}}(k)$-bandwidth property. Moreover, $\sum_{W \in \mathcal{W}} |\operatorname{out}(W)| \leq |\operatorname{out}(S)|(1 + \frac{1}{64\gamma_{\text{CMG}}(k)})$.*

The grouping technique. The grouping technique was first introduced by Chekuri, Khanna, and Shepherd [20] and has since been widely used in algorithms for network routing [17, 54, 1] to boost network connectivity and well-linkedness parameters. We summarize it in the following theorem, whose proof appears in Appendix A for completeness.

THEOREM 2.10. *Suppose we are given a connected graph $G = (V, E)$, with weights $w(v)$ on vertices $v \in V$, and a parameter p. Assume further that for each $v \in V$, $0 \leq w(v) \leq p$, and $\sum_{v \in V} w(v) \geq p$. Then we can find a partition $\mathcal{G}$ of the vertices in V, and for each group $U \in \mathcal{G}$, find a tree $T_U \subseteq G$ containing all vertices of U such that the trees $\{T_U\}_{U \in \mathcal{G}}$ are edge-disjoint, and for each $U \in \mathcal{G}$, $p \leq \sum_{v \in U} w(v) \leq 3p$.*

We will sometimes use the grouping theorem in slightly different settings. The first such setting is when we are given a subset $\mathcal{T} \subseteq V$ of vertices called terminals, and we would like to group them into groups of cardinality at least p and at most $3p$. In this case we will think of all nonterminal vertices as having weight 0, and terminal vertices as having weight 1. Instead of finding a partition $\mathcal{G}$ of all vertices, we will be looking for a partition $\mathcal{G}'$ of the set $\mathcal{T}$ of terminals. This partition is obtained from $\mathcal{G}$ by ignoring the nonterminal vertices. Another setting is when we are given a subset $E' \subseteq E$ of edges, and we would like to find a partition $\mathcal{G}$ of these edges into groups of at least p and at most $3p$ edges. As before, we would also like to find, for each group $U \in \mathcal{G}$, a tree T_U containing all edges of U, and we require that the trees $\{T_U\}_{U \in \mathcal{G}}$ are edge-disjoint. This setting can be reduced to the previous one by subdividing each edge $e \in E'$ with a terminal vertex. It is easy to verify that Theorem 2.10 can be applied in this setting as well.

3. The algorithm. This section is dedicated to proving Theorem 1.1. Our starting point is similar to that used in previous work on the problem [20, 17, 54, 1]: namely, we use the standard multicommodity flow LP relaxation for the EDP problem to partition our graph into several disjoint subgraphs and compute a subset of the demand pairs for each such subgraph, such that the terminals participating in the demand pairs are well-linked in the subgraph. We then solve the problem separately on each such subgraph. We use the standard multicommodity flow LP relaxation for EDP, where for each $1 \leq i \leq k$, we have an indicator variable x_i for whether or not we route the pair (s_i, t_i). Let $\mathcal{P}_i$ denote the set of all paths connecting s_i to t_i in G.

The LP relaxation is defined as follows:

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

While this LP has exponentially many variables, it can be efficiently solved using standard techniques, e.g., by using an equivalent polynomial-size edge-based LP formulation. Throughout the rest of the paper, we denote by OPT the value of the optimal solution to the LP. Clearly, the value of the optimal solution to the EDP problem instance is at most OPT. The next theorem follows from the work of Chekuri, Khanna, and Shepherd [20, 17], and we provide a short proof sketch in Appendix B for completeness.

THEOREM 3.1. *Suppose we are given a graph $G = (V, E)$ and a set $\mathcal{M}$ of k source-sink pairs in G. Then we can efficiently partition G into a collection $G_1, \ldots, G_\ell$ of vertex-disjoint induced subgraphs and compute, for each $1 \leq i \leq \ell$, a collection $\mathcal{M}_i \subseteq \mathcal{M}$ of source-sink pairs contained in G_i such that $\sum_{i=1}^{\ell} |\mathcal{M}_i| = \Omega(\mathsf{OPT}/\log^2 k)$, and, moreover, if for each $1 \leq i \leq \ell$, $\mathcal{T}_i$ denotes the set of terminals participating in pairs in $\mathcal{M}_i$, then $\mathcal{T}_i \subseteq V(G_i)$, and $\mathcal{T}_i$ is $\frac{1}{2}$-well-linked in G_i.*

We now proceed to solve the problem on each one of the graphs G_i separately. We assume without loss of generality that G_i is connected. In order to simplify the notation, we denote the graph G_i by G, the set $\mathcal{M}_i$ of the source-sink pairs by $\mathcal{M}$, and the set $\mathcal{T}_i$ of terminals by $\mathcal{T}$. For simplicity, we denote $|\mathcal{M}| = k$. Recall that $\mathcal{T}$ is $\frac{1}{2}$-well-linked in G, the degree of every terminal in $\mathcal{T}$ is 1, and the maximum vertex degree in G is at most 4. It is now enough to prove that we can route $\Omega(\frac{k}{\log^{20.5} k \log\log k})$ demand pairs in $\mathcal{M}$ with congestion at most 14. We also assume that $k > k_0$, where k_0 is a large enough constant: otherwise, we can simply pick any source-sink pair $(s, t) \in \mathcal{M}$, connect it with any path P, and output this as a solution. In particular, we will assume that $k > \log^{24} k$, and $\gamma_{\text{CMG}}(k) = \Theta(\log^2 k) > 20$.

Throughout the algorithm, we let $\gamma = \gamma_{\text{CMG}}(k) = \Theta(\log^2 k)$, and we use a parameter $k_1 = \frac{k}{192\gamma^3 \log\gamma} = \Omega(\frac{k}{\log^6 k \log\log k})$. We say that a cluster $C \subseteq V(G)$ is *small* if $|\operatorname{out}_G(C)| \leq k_1$, and we say that it is large otherwise.

Families of good routers. We define a family of good routers in graph G. We then proceed in two steps. First, we show that we can efficiently find a family of good routers. Next, we show that given such a family, we can find the desired routing of a subset of the source-sink pairs in $\mathcal{M}$.

DEFINITION 3.2. *We say that a subset $S \subseteq V(G) \setminus \mathcal{T}$ of vertices is a* good router *iff there is a subset $\Gamma \subseteq \operatorname{out}_G(S)$ of edges, with $|\Gamma| = k_1$, such that the following hold:*

- *S has the $\alpha_{\text{BW}}(k)$-bandwidth property for Γ. That is, for any partition (X, Y) of S, if $\Gamma_X = \Gamma \cap \operatorname{out}(X)$ and $\Gamma_Y = \Gamma \cap \operatorname{out}(Y)$, then $|E_G(X, Y)| \geq \alpha_{\text{BW}}(k) \cdot \min\{|\Gamma_X|, |\Gamma_Y|\}$.*

- *There is a flow F in graph G, where every edge $e \in \Gamma$ sends one flow unit to a distinct terminal $t_e \in \mathcal{T}$ (so for $e \neq e'$, $t_e \neq t_{e'}$), and the congestion caused by F is at most $2\beta(k)/\alpha_{\text{BW}}(k) = O(\log^{4.5} k)$.*

We say that a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of $\gamma = \gamma_{\text{CMG}}(k) = \Theta(\log^2 k)$ subsets of vertices is a family of good routers iff each subset S_j is a good router, and $S_1, \ldots, S_\gamma$ are pairwise disjoint.

We view the subset $\Gamma \subseteq \text{out}_G(S)$ of edges as part of the definition of a good router. In particular, when we say that we are given a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers, we assume that we are also given the corresponding subsets $\Gamma_j \subseteq \text{out}_G(S_j)$ of edges for all $1 \leq j \leq \gamma$. The rest of the proof consists of two parts. First, we show how to find a family of good routers in G. Next, we exploit this family to embed an expander into G and find the desired routing.

3.1. Finding a family of good routers. The goal of this section is to prove the following theorem.

THEOREM 3.3. *There is an efficient randomized algorithm that with high probability computes a family of good routers in graph G.*

In order to prove Theorem 3.3, we use the notions of *good clustering* and *legal contracted graphs*. Intuitively, these notions allow us to "hide" some irrelevant parts of our graph by contracting some connected clusters into supernodes. Our algorithm will perform a number of iterations. In every iteration, we start from some legal contracted graph and either produce a family of good routers or contract the graph even further. This guarantees that after polynomially many iterations, we will produce a family of good routers. We now turn to define the notions of good clustering and legal contracted graph.

We say that a partition $\mathcal{C}$ of $V(G)$ is a *good clustering* iff the following hold:

- Each terminal $t \in \mathcal{T}$ belongs to a cluster that contains only t. That is, for each $t \in \mathcal{T}$, $\{t\} \in \mathcal{C}$.
- For each cluster $C \in \mathcal{C}$, $|\text{out}_G(C)| \leq k_1$, and the set C has the $\alpha_{\text{BW}}(k)$-bandwidth property. (In particular, since G is connected, due to the bandwidth property, so is $G[C]$.)

Given a good clustering $\mathcal{C}$, we let $G' = G_{\mathcal{C}}$ be obtained from G by contracting each cluster $C \in \mathcal{C}$ into a supernode v_C. We delete self-loops but keep parallel edges. A graph G' obtained in this manner from a good clustering $\mathcal{C}$ is called a *legal contracted graph*.

Notice that graph G' may contain parallel edges, and $\mathcal{T}$ remains $\frac{1}{2}$-well-linked in G'. Also, since the maximum vertex degree in G is 4, the maximum vertex degree in G' is at most k_1, and every terminal has degree 1. Every edge in graph G' corresponds to some edge in the original graph G, and we will not distinguish between them. In particular, for every vertex subset $S' \subseteq V(G')$, if $S \subseteq V(G)$ is the corresponding subset of vertices in G, where every supernode $v_C \in S'$ is replaced by the vertices of C, then there is a one-to-one mapping between $\text{out}_{G'}(S')$ and $\text{out}_G(S)$, and we will identify the edges in these two sets, that is, $\text{out}_{G'}(S') = \text{out}_G(S)$. We need the following simple claim.

CLAIM 3.4. *If G' is a legal contracted graph for G, then $G' \setminus \mathcal{T}$ contains at least $k/6$ edges.*

Proof. For each terminal $t \in \mathcal{T}$, let e_t be the unique edge adjacent to t in G', and let u_t be the other endpoint of e_t. We partition the terminals in $\mathcal{T}$ into groups, where two terminals t, t' belong to the same group iff $u_t = u_{t'}$. Let $\mathcal{G}$ be the resulting partition of the terminals. Since the degree of every vertex in G' is at most k_1, each

group $U \in \mathcal{G}$ contains at most k_1 terminals. Next, we partition the terminals in $\mathcal{T}$ into two subsets X, Y, where $|X|, |Y| \geq k/3$, and for each group $U \in \mathcal{G}$, either $U \subseteq X$ or $U \subseteq Y$ holds. It is possible to find such a partition by greedily processing each group $U \in \mathcal{G}$ and adding all terminals of U to one of the subsets X or Y that currently contains fewer terminals. Finally, we remove terminals from set X until $|X| = k/3$, and we do the same for Y. Since the terminals are $\frac{1}{2}$-well-linked in G', it is possible to route $k/3$ flow units from the terminals in X to the terminals in Y, with edge-congestion at most 2. Since no group U is split between the two sets X and Y, each flow-path must contain at least one edge of $G' \setminus \mathcal{T}$. Therefore, the number of edges in $G' \setminus \mathcal{T}$ is at least $k/6$. □

In order to prove Theorem 3.3, we use the following theorem.

THEOREM 3.5. *Let $\mathcal{C}$ be a good clustering of G. Then there is an efficient randomized algorithm that with high probability either returns a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers in G, together with the corresponding subsets $\Gamma_j \subseteq \mathrm{out}_G(S_j)$ of edges for all $1 \leq j \leq \gamma$, or finds a good clustering $\mathcal{C}'$ of G, with $\sum_{C \in \mathcal{C}'} |\mathrm{out}_G(C)| < \sum_{C \in \mathcal{C}} |\mathrm{out}_G(C)|$.*

We prove Theorem 3.5 below and complete the proof of Theorem 3.3 here first. We start with the good clustering $\mathcal{C} = \{\{v\} \mid v \in V(G)\}$, where every vertex belongs to a separate cluster, and repeatedly apply Theorem 3.5 to it. In every iteration, if Theorem 3.5 returns a family $\mathcal{F}$ of good routers, then we terminate the algorithm and return $\mathcal{F}$. Otherwise, we replace $\mathcal{C}$ with $\mathcal{C}'$ and continue to the next iteration. Clearly, after at most $2|E(G)|$ iterations, the algorithm will produce a family of good routers with high probability. From now on we focus on the proof of Theorem 3.5.

Let $G' = G_{\mathcal{C}}$ be the legal contracted graph corresponding to $\mathcal{C}$, and let m be the number of edges in $G' \setminus \mathcal{T}$. From Claim 3.4, $m \geq k/6$. The proof consists of two steps. First, we randomly partition the vertices in $G' \setminus \mathcal{T}$ into γ subsets $X_1, \ldots, X_\gamma$. We show that with high probability, for each $1 \leq j \leq \gamma$, $|\mathrm{out}_{G'}(X_j)| < \frac{10m}{\gamma}$, while the number of edges with both endpoints in X_j, $|E_{G'}(X_j)| \geq \frac{m}{2\gamma^2}$. Therefore, $|E_{G'}(X_j)| > \frac{|\mathrm{out}_{G'}(X_j)|}{20\gamma}$ with high probability. For each $j : 1 \leq j \leq \gamma$, we then try to recover a good router S_j from the cluster X_j. If we succeed, then we obtain a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers. If we fail to recover a good router for some $1 \leq j \leq \gamma$, then we will produce a new good clustering $\mathcal{C}'$, with $\sum_{C \in \mathcal{C}'} |\mathrm{out}_G(C)| < \sum_{C \in \mathcal{C}} |\mathrm{out}_G(C)|$.

We start with the first step. We partition the vertices in $V(G') \setminus \mathcal{T}$ into 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.

CLAIM 3.6. *With probability at least $\frac{1}{2}$, for each $1 \leq j \leq \gamma$, $|\mathrm{out}_{G'}(X_j)| < \frac{10m}{\gamma}$, while $|E_{G'}(X_j)| \geq \frac{m}{2\gamma^2}$.*

Proof. Let $H = G' \setminus \mathcal{T}$. Fix some $1 \leq j \leq \gamma$. Let $\mathcal{E}_1(j)$ be the bad event that $\sum_{v \in X_j} d_H(v) \geq \frac{2m}{\gamma} \cdot (1 + \frac{1}{\gamma})$. In order to bound the probability of $\mathcal{E}_1(j)$, we define, for each vertex $v \in V(H)$, a random variable x_v, whose value is $\frac{d_H(v)}{k_1}$ if $v \in X_j$ and 0 otherwise. Notice that $x_v \in [0, 1]$, and the random variables $\{x_v\}_{v \in V(H)}$ are pairwise independent. Let $B = \sum_{v \in V(H)} x_v$. Then the expectation of B, $\mu_1 = \sum_{v \in V(H)} \frac{d_H(v)}{\gamma k_1} = \frac{2m}{\gamma k_1}$. Using the standard Chernoff bound (see, e.g., Theorem 1.1 in [25]),

$$\mathbf{Pr}\left[\mathcal{E}_1(j)\right] = \mathbf{Pr}\left[B > (1 + 1/\gamma)\,\mu_1\right] \leq e^{-\mu_1/(3\gamma^2)} = e^{-\frac{2m}{3\gamma^3 k_1}} < \frac{1}{6\gamma}$$

since $m \geq k/6$ and $k_1 = \frac{k}{192\gamma^3 \log \gamma}$.

For each terminal $t \in \mathcal{T}$, let e_t be the unique edge adjacent to t in graph G', and let u_t be its other endpoint. Let $U = \{u_t \mid t \in \mathcal{T}\}$. For each vertex $u \in U$, let $w(u)$ be the number of terminals t, such that $u = u_t$. Notice that $w(u) \leq k_1$ must hold. We say that a bad event $\mathcal{E}_2(j)$ happens iff $\sum_{u \in U \cap X_j} w(u) \geq \frac{k}{\gamma} \cdot (1 + \frac{1}{\gamma})$. In order to bound the probability of the event $\mathcal{E}_2(j)$, we define, for each $u \in U$, a random variable y_u, whose value is $w(u)/k_1$ if $u \in X_j$ and 0 otherwise. Notice that $y_u \in [0, 1]$, and the variables y_u are independent for all $u \in U$. Let $Y = \sum_{u \in U} y_u$. The expectation of Y is $\mu_2 = \frac{k}{k_1 \gamma}$, and event $\mathcal{E}_2(j)$ holds iff $Y \geq \frac{k}{k_1 \gamma} \cdot (1 + \frac{1}{\gamma}) \geq \mu_2 \cdot (1 + \frac{1}{\gamma})$. Using the standard Chernoff bound again, we get that

$$\mathbf{Pr}\left[\mathcal{E}_2(j)\right] \leq e^{-\mu_2/(3\gamma^2)} \leq e^{-k/(3k_1\gamma^3)} \leq \frac{1}{6\gamma}$$

since $k_1 = \frac{k}{192\gamma^3 \log \gamma}$. Notice that if events $\mathcal{E}_1(j), \mathcal{E}_2(j)$ do not happen, then

$$|\operatorname{out}_{G'}(X_j)| \leq \sum_{v \in X_j} d_H(v) + \sum_{u \in U \cap X_j} w(u) \leq \left(1 + \frac{1}{\gamma}\right)\left(\frac{2m}{\gamma} + \frac{k}{\gamma}\right) < \frac{10m}{\gamma}$$

since $m \geq k/6$.

Let $\mathcal{E}_3(j)$ be the bad event that $|E_{G'}(X_j)| < \frac{m}{2\gamma^2}$. We next prove that $\mathbf{Pr}\left[\mathcal{E}_3(j)\right] \leq \frac{1}{6\gamma}$. We say that two edges $e, e' \in E(G' \setminus \mathcal{T})$ are *independent* iff they do not share any endpoints. Our first step is to compute a partition $U_1, \ldots, U_r$ of the set $E(G' \setminus \mathcal{T})$ of edges, where $r \leq 2k_1$, such that for each $1 \leq i \leq r$, $|U_i| \geq \frac{m}{4k_1}$, and all edges in set U_i are mutually independent. In order to compute such a partition, we construct an auxiliary graph Z, whose vertex set is $\{v_e \mid e \in E(H)\}$, and there is an edge $(v_e, v_{e'})$ iff e and e' are not independent. Since the maximum vertex degree in G' is at most k_1, the maximum vertex degree in Z is bounded by $2k_1 - 2$. We use the following theorem due to Hajnal and Szemerédi [33].

THEOREM 3.7 (see [33]). *For any graph H with maximum vertex degree Δ, there is a partition $S_1, \ldots, S_r$ of $V(H)$ into $r \leq \Delta + 1$ disjoint subsets, such that for each $1 \leq i \leq r$, S_i is an independent set, and for $1 \leq i \neq i' \leq r$, $||S_i| - |S_{i'}|| \leq 1$.*

From Theorem 3.7, we can find a partition $V_1, \ldots, V_r$ of the vertices of Z into $r \leq 2k_1$ subsets, where each subset V_i is an independent set, and $|V_i| \geq \frac{|V(Z)|}{r} - 1 \geq \frac{m}{4k_1}$. The partition $V_1, \ldots, V_r$ of the vertices of Z gives the desired partition $U_1, \ldots, U_r$ of the edges of $G' \setminus \mathcal{T}$. For each $1 \leq i \leq r$, we say that the bad event $\mathcal{E}_3^i(j)$ happens iff $|U_i \cap E(X_j)| < \frac{|U_i|}{2\gamma^2}$. Notice that if $\mathcal{E}_3(j)$ happens, then event $\mathcal{E}_3^i(j)$ must happen for some $1 \leq i \leq r$. Fix some $1 \leq i \leq r$. The expectation of $|U_i \cap E(X_j)|$ is $\mu_3 = \frac{|U_i|}{\gamma^2}$. Since all edges in U_i are independent, we can use the standard Chernoff bound to bound the probability of $\mathcal{E}_3^i(j)$ as follows:

$$\mathbf{Pr}\left[\mathcal{E}_3^i(j)\right] = \mathbf{Pr}\left[|U_i \cap E(X_j)| < \mu_3/2\right] \leq e^{-\mu_3/8} = e^{-\frac{|U_i|}{8\gamma^2}}.$$

Since $|U_i| \geq \frac{m}{4k_1}$, $m \geq k/6$, $k_1 = \frac{k}{192\gamma^3 \log \gamma}$, and $\gamma = \Theta(\log^2 k)$, this is bounded by $\frac{1}{12k_1\gamma}$. We conclude that $\mathbf{Pr}\left[\mathcal{E}_3^i(j)\right] \leq \frac{1}{12k_1\gamma}$, and by using the union bound over all $1 \leq i \leq r$, $\mathbf{Pr}\left[\mathcal{E}_3(j)\right] \leq \frac{1}{6\gamma}$.

Using the union bound over all $1 \leq j \leq \gamma$, with probability at least $\frac{1}{2}$, none of the events $\mathcal{E}_1(j), \mathcal{E}_2(j), \mathcal{E}_3(j)$ for any $1 \leq j \leq \gamma$ happen, and so for each $1 \leq j \leq \gamma$, $|\operatorname{out}_{G'}(X_j)| < \frac{10m}{\gamma}$, and $|E_{G'}(X_j)| \geq \frac{m}{2\gamma^2}$ must hold. ∎

Given a partition $X_1, \ldots, X_\gamma$, we can efficiently check whether the conditions of Claim 3.6 hold. If they do not hold, we repeat the randomized partitioning procedure. From Claim 3.6, we are guaranteed that with high probability, after $\text{poly}(n)$ iterations, we will obtain a partition with the desired properties. Assume now that we are given the partition $X_1, \ldots, X_\gamma$ of $V(G') \setminus \mathcal{T}$, for which the conditions of Claim 3.6 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 uncontract each cluster, that is, $X'_j = \bigcup_{v_C \in X_j} C$. Notice that $\{X'_j\}_{j=1}^{\gamma}$ is a partition of $V(G) \setminus \mathcal{T}$. We use the following theorem to finish the proof of Theorem 3.5.

THEOREM 3.8. *There is an efficient algorithm that, for each $1 \leq j \leq \gamma$, either finds a good router $S_j \subseteq X'_j$ or produces a good clustering $\mathcal{C}'$ of G, with $\sum_{C \in \mathcal{C}'} |\operatorname{out}_G(C)| < \sum_{C \in \mathcal{C}} |\operatorname{out}_G(C)|$.*

In order to complete the proof of Theorem 3.5, we apply Theorem 3.8 to each cluster X_j in turn. If, for some $1 \leq j \leq \gamma$, Theorem 3.8 produces a good clustering $\mathcal{C}'$, with $\sum_{C \in \mathcal{C}'} |\operatorname{out}_G(C)| < \sum_{C \in \mathcal{C}} |\operatorname{out}_G(C)|$, then we are done. Otherwise, for each $1 \leq j \leq \gamma$, we obtain a good router $S_j \subseteq X'_j$, and, since the sets $X'_1, \ldots, X'_\gamma$ are mutually disjoint, we obtain a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers. In order to complete the proof of Theorem 3.5, it now remains to prove Theorem 3.8.

Proof of Theorem 3.8. Recall that we say that a cluster $S \subseteq V(G)$ is large if $|\operatorname{out}_G(S)| > k_1$, and we say that it is small otherwise. Fix some $1 \leq j \leq \gamma$. We partition the set $\mathcal{C}$ of clusters into two subsets: $\mathcal{W}_1$ contains all clusters C with $C \cap X'_j = \emptyset$, and $\mathcal{W}_2$ contains all clusters $C \subseteq X'_j$. Throughout the algorithm, we also maintain another partition $\tilde{\mathcal{C}}$ of $V(G)$ into clusters (which may not be a good clustering). We will also maintain a partition $(\tilde{\mathcal{W}}_1, \tilde{\mathcal{W}}_2)$ of $\tilde{\mathcal{C}}$, where $\tilde{\mathcal{W}}_1 = \tilde{\mathcal{C}} \cap \mathcal{W}_1$, and $\tilde{\mathcal{W}}_2 = \tilde{\mathcal{C}} \setminus \tilde{\mathcal{W}}_1$. Recall that from Claim 3.6, $\sum_{C \in \mathcal{W}_2} |\operatorname{out}_G(C)| \geq |\operatorname{out}_G(X'_j)| + 2|E_{G'}(X_j)| > |\operatorname{out}_G(X'_j)|(1 + \frac{1}{10\gamma})$. Our goal is to either find a good router $S_j \subseteq X'_j$ or produce a good clustering $\tilde{\mathcal{C}}$, where $\sum_{C \in \tilde{\mathcal{W}}_2} |\operatorname{out}_G(C)| < |\operatorname{out}_G(X'_j)|(1 + \frac{1}{10\gamma})$. This will ensure that $\sum_{C \in \tilde{\mathcal{C}}} |\operatorname{out}(C)| < \sum_{C \in \mathcal{C}} |\operatorname{out}(C)|$.

Assume now that we are given some large cluster $C \subseteq V(G) \setminus \mathcal{T}$ and a current partition $\tilde{\mathcal{C}}$ of $V(G)$ into clusters. We say that a partition (A, B) of $V(G)$ is a *small canonical separator for C* iff $C \subseteq A$, $\mathcal{T} \subseteq B$, $|E_G(A, B)| \leq k_1$, and, for each cluster $C' \in \tilde{\mathcal{C}}$, either $C' \subseteq A$ or $C' \subseteq B$ holds.

Our algorithm proceeds in two steps. In the first step, we either find a good router $S_j \subseteq X'_j$ or compute a partition $\tilde{\mathcal{C}}$ of $V(G)$ into clusters, such that each large cluster $C \in \tilde{\mathcal{C}}$ has a small canonical separator. If we find a good router $S_j \subseteq X'_j$, then we terminate the algorithm and return S_j. Otherwise, we continue to the second step, which transforms the partition $\tilde{\mathcal{C}}$ into a good clustering.

Step 1. Given a partition $\tilde{\mathcal{C}}$ of $V(G)$, a cluster $C \in \tilde{\mathcal{C}}$ is called a *candidate cluster* if it is a large cluster, and there is no small canonical separator (A, B) for C. Notice that, in this case, there is a flow F in the corresponding contracted graph $G_{\tilde{\mathcal{C}}}$, of value at least k_1, from v_C to $\mathcal{T}$. This step is summarized in the following claim.

CLAIM 3.9. *There is an efficient algorithm that either finds a good router $S_j \subseteq X'_j$ or computes a partition $\mathcal{R}$ of X'_j, such that $\sum_{C \in \mathcal{R}} |\operatorname{out}_G(C)| \leq |\operatorname{out}_G(X'_j)|(1 + \frac{1}{64\gamma})$, and, if $\tilde{\mathcal{C}} = \mathcal{R} \cup \mathcal{W}_1$, then no cluster of $\tilde{\mathcal{C}}$ is a candidate cluster.*

Proof. We start with $\mathcal{R} = \{X'_j\}$. Throughout the algorithm, we maintain a partition $\tilde{\mathcal{C}}$ of $V(G)$ into clusters, where $\tilde{\mathcal{C}} = \mathcal{R} \cup \mathcal{W}_1$. While $\tilde{\mathcal{C}}$ contains at least one candidate cluster, let S be any such cluster. Notice that $S \in \mathcal{R}$ must hold, as every

cluster in $\mathcal{W}_1$ is small. For simplicity, we denote $G_{\tilde{\mathcal{C}}}$ by $\tilde{G}$. From the integrality of flow, there is a collection $\mathcal{P}$ of k_1 edge-disjoint paths in $\tilde{G}$ connecting distinct edges in $\mathrm{out}_{\tilde{G}}(S)$ to distinct terminals in $\mathcal{T}$. Let $\Gamma \subseteq \mathrm{out}_{\tilde{G}}(S)$ be the set of k_1 edges which serve as endpoints of the paths in $\mathcal{P}$. We set up an instance of the sparsest cut problem in graph $G[S] \cup \mathrm{out}_G(S)$, where the edges in set Γ serve as terminals. We then run the algorithm $\mathcal{A}_{\mathrm{ARV}}$ on the resulting instance. If the algorithm returns a cut (Y, Z) of sparsity less than $\alpha(k)$, then (Y, Z) is a $(k, \alpha(k))$-violating cut for S. We then replace S with Y and Z in $\mathcal{R}$, and update $\tilde{\mathcal{C}}$ and $\tilde{G}$ accordingly. This ends the current iteration, and we then proceed to the next iteration. Assume now that algorithm $\mathcal{A}_{\mathrm{ARV}}$ returns a cut whose sparsity is at least $\alpha(k)$. Then we are guaranteed that S has the $\alpha_{\mathrm{BW}}(k) = \alpha(k)/\alpha_{\mathrm{ARV}}(k)$-bandwidth property for Γ. Recall that we are given a set $\mathcal{P}$ of k_1 edge-disjoint paths connecting the edges in Γ to the terminals $\mathcal{T}$ in graph $\tilde{G}$, where each path connects a distinct edge $e \in \Gamma$ to a distinct terminal $t_e \in \mathcal{T}$. In order for S to be a good router, a low-congestion flow connecting the edges in Γ to the terminals must exist in the original graph G. We will try to find this flow, as follows. The flow will follow the paths in $\mathcal{P}$, except that we need to specify how the flow is routed inside each cluster C for $C \in \tilde{\mathcal{C}}$. Observe that for each such cluster C, the paths in $\mathcal{P}$ define a set D_C of 1-restricted demands on the edges of $\mathrm{out}_G(C)$. Moreover, the total number of edges in $\mathrm{out}_G(C)$ participating in the paths in $\mathcal{P}$ is at most k_1, as there are only k_1 paths in $\mathcal{P}$ and we can assume without loss of generality that they are simple. If $C \in \mathcal{W}_1$, then we are guaranteed that cluster C has the $\alpha_{\mathrm{BW}}(k)$-bandwidth property in graph G. From Observation 2.7, we can route the set D_C of demands inside $G[C]$ with congestion at most $2\beta(k)/\alpha_{\mathrm{BW}}(k)$. Otherwise, $C \in \mathcal{R}$, and it is possible that we cannot route the set D_C of demands inside $G[C]$ with congestion at most $2\beta(k)/\alpha_{\mathrm{BW}}(k)$. We then proceed as follows. If, for each cluster $C \in \mathcal{R}$, we can route the set D_C of demands inside $G[C]$ with congestion at most $2\beta(k)/\alpha_{\mathrm{BW}}(k)$, then S is a good router. We terminate the algorithm and return S. Otherwise, let $C \in \mathcal{R}$ be any cluster, for which such flow does not exist. Consider the instance of the sparsest cut problem defined on the graph $G[C] \cup \mathrm{out}_G(C)$, where the edges of $\mathrm{out}_G(C)$ with nonzero demand serve as terminals (recall that there are at most $2k_1$ such edges). Then the value of the sparsest cut in this instance is at most $\alpha_{\mathrm{BW}}(k)$, and so by applying algorithm $\mathcal{A}_{\mathrm{ARV}}$ to this instance of sparsest cut, we will obtain a $(k, \alpha(k))$-violating cut (Y, Z) for set C. We then remove C from $\mathcal{R}$, and add Y and Z to $\tilde{\mathcal{C}}$ instead. We also update $\tilde{\mathcal{C}}$ and $\tilde{G}$ accordingly and end the current iteration. This concludes the description of the algorithm. Notice that the final set $\mathcal{R}$ was obtained from X'_j by running the bandwidth-decomposition algorithm from Theorem 2.8 on X'_j, as in each step we computed a (k, α)-violating cut of some cluster in the current partition $\mathcal{R}$ of X'_j. Therefore, from Theorem 2.8, $\sum_{C \in \mathcal{R}} |\mathrm{out}_G(C)| \leq |\mathrm{out}_G(X'_j)|(1 + \frac{1}{64\gamma})$. If the algorithm does not terminate with a good router $S \subseteq X'_j$, then it is guaranteed to terminate after $O(|V(G)|)$ iterations, with a partition $\mathcal{R}$ of X'_j, and a corresponding partition $\tilde{\mathcal{C}}$ of $V(G)$, such that $\tilde{\mathcal{C}}$ contains no candidate clusters. □

If the algorithm from Claim 3.9 returns a good router $S \subseteq X'_j$, then we terminate the algorithm and return S. We assume from now on that Claim 3.9 returns a partition $\mathcal{R}$ of X'_j into clusters and a corresponding partition $\tilde{\mathcal{C}}$ of $V(G)$, where $\tilde{\mathcal{C}} = \mathcal{R} \cup \mathcal{W}_1$, such that no cluster of $\tilde{\mathcal{C}}$ is a candidate cluster, and $\sum_{C \in \mathcal{R}} |\mathrm{out}_G(C)| \leq |\mathrm{out}_G(X'_j)|(1 + \frac{1}{64\gamma})$.

Step 2. In this step, we transform the partition $\tilde{\mathcal{C}}$ of $V(G)$ computed in Step 1

into a good clustering. We start with the following claim.

CLAIM 3.10. *Let $\tilde{\mathcal{C}}$ be the partition of $V(G)$ computed in Step 1. Then there is an efficient algorithm to find a collection $\mathcal{S}$ of disjoint small clusters, such that each cluster $S \in \mathcal{S}$ is contained in $V(G) \setminus \mathcal{T}$, and for each large cluster $C \in \tilde{\mathcal{C}}$, there is some cluster $S \in \mathcal{S}$ with $C \subseteq \mathcal{S}$. Moreover, for every pair $C' \in \tilde{\mathcal{C}}$ and $S \in \mathcal{S}$ of clusters, either $C' \subseteq S$ or $C' \cap S = \emptyset$.*

Proof. Throughout the algorithm, we maintain a collection $\mathcal{S}$ of disjoint subsets of nonterminal vertices of G, where each set $S \in \mathcal{S}$ is a small cluster. We will ensure throughout the algorithm that for each pair $C' \in \tilde{\mathcal{C}}$ and $S \in \mathcal{S}$ of clusters, either $C' \subseteq S$ or $C' \cap S = \emptyset$. We say that a large cluster $C \in \tilde{\mathcal{C}}$ is *covered* by $\mathcal{S}$ iff there is a cluster $S \in \mathcal{S}$ with $C \subseteq \mathcal{S}$. We start with $\mathcal{S} = \emptyset$.

While there is a large cluster in $\tilde{\mathcal{C}}$ that is not covered by $\mathcal{S}$, let C be any such cluster. Recall that there is a small canonical separator (A, B) for C in G. Let $\mathcal{L} \subseteq \tilde{\mathcal{C}}$ be the set of large clusters covered by $\mathcal{S} \cup \{A\}$. We now consider clusters $S \in \mathcal{S}$ one by one. For each such cluster S, if $S \setminus A$ is a small cluster, then we replace S with $S \setminus A$ in $\mathcal{S}$. Notice that all clusters of $\mathcal{L}$ remain covered by the new set $\mathcal{S} \cup \{A\}$. Otherwise, if $S \setminus A$ is a large cluster, then, from the submodularity of cuts, $|\operatorname{out}(S)| + |\operatorname{out}(A)| \geq |\operatorname{out}(S \setminus A)| + |\operatorname{out}(A \setminus S)|$, and since S and A are small clusters, $A \setminus S$ is a small cluster. We replace A with $A \setminus S$ and continue. Notice that all clusters of $\mathcal{L}$ remain covered by the new set $\mathcal{S} \cup \{A\}$. Once all clusters of $\mathcal{S}$ are processed, we add the final set A to $\mathcal{S}$ and continue to the next iteration. It is easy to see that in each iteration, the number of large clusters in $\tilde{\mathcal{C}}$ that are not covered by $\mathcal{S}$ decreases by at least 1, and we maintain the property that for each pair $C' \in \tilde{\mathcal{C}}$ and $S \in \mathcal{S}$ of clusters, either $C' \subseteq S$ or $C' \cap S = \emptyset$. Once all large clusters of $\tilde{\mathcal{C}}$ are covered, we obtain the desired set $\mathcal{S}$ of small clusters. ∎

We assume without loss of generality that for each cluster $S \in \mathcal{S}$, there is some large cluster $C_S \in \tilde{\mathcal{C}}$ with $C_S \subseteq S$, since otherwise we can remove S from $\mathcal{S}$. Let $\tilde{\mathcal{C}}'$ be a partition of $V(G)$, obtained from $\tilde{\mathcal{C}}$ as follows: for each cluster $S \in \mathcal{S}$, we delete all clusters $C \subseteq S$ from $\tilde{\mathcal{C}}$, and we add S to $\tilde{\mathcal{C}}'$ instead. Notice that every cluster in $\tilde{\mathcal{C}}'$ is now a small cluster. We let $\tilde{\mathcal{W}}_1 = \tilde{\mathcal{C}}' \cap \mathcal{W}_1$ and $\tilde{\mathcal{W}}_2 = \tilde{\mathcal{C}}' \setminus \mathcal{W}_1$. Notice that for every cluster $S \in \tilde{\mathcal{W}}_2$, either $S \in \mathcal{R}$ or there is some large cluster $C_S \in \mathcal{R}$, such that $C_S \subseteq S$. In the latter case, since C_S is a large cluster, and S is a small cluster, $|\operatorname{out}_G(S)| < |\operatorname{out}_G(C_S)|$. Therefore,

$$\sum_{S \in \tilde{\mathcal{W}}_2} |\operatorname{out}_G(S)| \leq \sum_{C \in \mathcal{R}} |\operatorname{out}_G(C)| \leq |\operatorname{out}_G(X'_j)| \left(1 + \frac{1}{64\gamma}\right).$$

Our final step is to compute, for each cluster $C \in \tilde{\mathcal{W}}_2$, a partition $\mathcal{W}(C)$ into small clusters that have the $\alpha_{\mathrm{BW}}(k)$-bandwidth property, such that $\sum_{C' \in \mathcal{W}(C)} |\operatorname{out}_G(C')| \leq |\operatorname{out}_G(C)|(1 + \frac{1}{64\gamma})$, using Corollary 2.9. Let $\mathcal{R}' = \bigcup_{C \in \tilde{\mathcal{W}}_2} \mathcal{W}(C)$. Then

$$\begin{aligned}\sum_{C' \in \mathcal{R}'} |\operatorname{out}_G(C')| &\leq \sum_{C \in \tilde{\mathcal{W}}_2} |\operatorname{out}_G(C)| \left(1 + \frac{1}{64\gamma}\right) \leq |\operatorname{out}_G(X'_j)| \left(1 + \frac{1}{64\gamma}\right)^2 \\ &< |\operatorname{out}_G(X'_j)| \left(1 + \frac{1}{20\gamma}\right).\end{aligned}$$

Our final good partition $\mathcal{C}'$ of $V(G)$ is the union of $\tilde{\mathcal{W}}_1$ and $\mathcal{R}'$. It is immediate

to verify that this is indeed a good partition. Moreover,

$$\begin{aligned}\sum_{C\in\mathcal{C}'}|\operatorname{out}_G(C)| &\le \sum_{C\in\tilde{\mathcal{W}}_1}|\operatorname{out}_G(C)| + \sum_{C\in\mathcal{R}'}|\operatorname{out}_G(C)|\\ &< \sum_{C\in\mathcal{W}_1}|\operatorname{out}_G(C)| + |\operatorname{out}_G(X'_j)|\left(1+\frac{1}{20\gamma}\right)\\ &< \sum_{C\in\mathcal{W}_1}|\operatorname{out}_G(C)| + \sum_{C\in\mathcal{W}_2}|\operatorname{out}_G(C)|\\ &= \sum_{C\in\mathcal{C}}|\operatorname{out}_G(C)|,\end{aligned}$$

since $\sum_{C\in\mathcal{W}_2}|\operatorname{out}_G(C)| = |\operatorname{out}_G(X'_j)| + 2|E_{G'}(X_j)| \ge |\operatorname{out}_G(X'_j)|(1+\frac{1}{10\gamma})$. □

3.2. Finding the routing. In this section, we exploit the family $\mathcal{F}$ of good routers, in order to find a routing of $\Omega(\frac{k}{\log^{20.5} k \log\log k})$ pairs in $\mathcal{M}$ with congestion at most 14.

We assume that we are given a family $\mathcal{F} = \{S_1,\ldots,S_\gamma\}$ of good routers in G. For each $1 \le j \le \gamma$, we are also given a subset $\Gamma_j \subseteq \operatorname{out}_G(S_j)$ of edges, such that S_j has the $\alpha_{\text{BW}}(k)$-bandwidth property for Γ_j, and there is a flow $F_j : \Gamma_j \rightsquigarrow_\eta \mathcal{T}$, where each edge $e \in \Gamma_j$ sends one flow unit to a distinct terminal t_e, and the total congestion due to F_j is at most $\eta = 2\beta(k)/\alpha_{\text{BW}}(k)$.

In order to find the final routing, we build an expander over a subset of terminals and embed it into graph G. More precisely, we select an arbitrary subset $\mathcal{M}' \subseteq \mathcal{M}$ of $k'/2$ source-sink pairs, for some $k' = k/\operatorname{poly}\log k$, whose precise value is defined later. Let $\mathcal{T}' \subseteq \mathcal{T}$ be the subset of terminals participating in pairs in $\mathcal{M}'$, and assume that $\mathcal{T}' = \{t_1,\ldots,t_{k'}\}$. We construct an expander X over the set $\{v_1,\ldots,v_{k'}\}$ of vertices, which is then embedded into the graph G as follows. For each $1 \le i \le k'$, we define a connected subgraph C_i of G that represents the vertex v_i of the expander. For each edge $e = (v_i, v_j) \in E(X)$, we define a path P_e connecting a vertex of C_i to a vertex of C_j in G. We will ensure that each edge of G may only appear in a small constant number of the subgraphs C_i and a small constant number of the paths P_e. We will also ensure that for each $1 \le i \le k'$, terminal $t_i \in C_i$. We will think about the expander vertex v_i as representing the terminal t_i. The idea is that any *vertex-disjoint* routing of the terminal pairs in the expander X can now be translated into a low edge-congestion routing in the original graph G.

We now turn to describe the construction of the expander X and the connected subgraphs $C_1,\ldots,C_{k'}$ that we use to embed X into G. The construction exploits the family $\mathcal{F} = \{S_1,\ldots,S_\gamma\}$ of good routers. We construct a collection $T_1,\ldots,T_{k'}$ of trees in graph G. Each such tree T_i contains, for each $1 \le j \le \gamma$, an edge $e_{i,j} \in \Gamma_j$. For each $1 \le j \le \gamma$, the edges $e_{1,j}, e_{2,j},\ldots,e_{k',j}$ are all distinct, and we think of the edge $e_{i,j}$ as the representative of the vertex $v_i \in V(X)$ for the set S_j. In other words, each tree T_i spans γ representatives of the vertex v_i: one representative $e_{i,j}$ for each set $S_j \in \mathcal{F}$. We will ensure that each edge of graph G only participates in a constant number of such trees. Additionally, we build a set $\mathcal{P} = \{P_t \mid t \in \mathcal{T}'\}$ of paths, where path P_t connects the terminal t to a distinct tree T_i (so if $t \ne t'$, then t and t' are connected to different trees), and the total congestion caused by paths in $\mathcal{P}$ is at most 4. We rename the terminals in $\mathcal{T}'$, so that t_i denotes the terminal that is connected to the tree T_i. The final subgraph C_i of G is simply the union of the tree T_i and the path P_{t_i}.

In order to construct the expander X over the set $\{v_1,\ldots,v_{k'}\}$ of vertices, we use the cut-matching game of [39], where we use the subgraph $G[S_j]$ of G to route the jth matching between the corresponding representatives $e_{1,j},e_{2,j},\ldots,e_{k',j}$ of the vertices $v_1,\ldots,v_{k'}$, respectively. Recall that we are only guaranteed that sets $\{S_j\}_{j=1}^{\gamma}$ have the $\alpha_{\text{BW}}(k)$-bandwidth property for the edges in Γ_j, and so in order to route these matchings, we may have to incur the congestion of $\Omega(1/\alpha_{\text{BW}}(k))$, which we cannot afford. However, this problem is easy to overcome by performing a suitable grouping of the edges of Γ_j.

The rest of the algorithm proceeds in three steps. In the first step, we perform groupings of the edges in the subsets Γ_j for $1\le j\le\gamma$. In the second step, we construct the trees $T_1,\ldots,T_{k'}$. In the third step, we finish the construction of the expander X and its embedding into G and produce the final routing of a subset of demand pairs in $\mathcal{M}'$.

Step 1*: Edge grouping.* In this step we compute, for each $1\le j\le\gamma$, a grouping of the edges in Γ_j. We then establish some properties of these groupings. We use the following two parameters: $p=8\beta(k)/\alpha_{\text{BW}}(k)=O(\log^{4.5}k)$ is the grouping parameter for the sets Γ_j. The second parameter, $k'=\lfloor\frac{1}{4\gamma^3}\cdot\lfloor\frac{k_1}{6p}\rfloor\rfloor=\Omega(\frac{k}{\log^{16.5}k\log\log k})$, is the number of the vertices in the expander X that we will eventually construct. We assume without loss of generality that k' is an even integer; otherwise we round it down to the closest even integer.

Fix some $1\le j\le\gamma$. Since $G[S_j]\cup\text{out}_G(S_j)$ is a connected graph, we can find a spanning tree T_j of this graph and partition the edges of Γ_j along this tree into groups whose size is at least p and at most $3p$ using Theorem 2.10. Let $\mathcal{G}_j$ be the resulting collection of groups, and let $k^*=\lfloor\frac{k_1}{6p}\rfloor$. For each group $U\in\mathcal{G}_j$, let $T_j(U)$ be the subtree of the tree T_j spanning the edges of U. For each group $U\in\mathcal{G}_j$, we select one arbitrary representative edge, and we let Γ'_j denote this set of representative edges. For each $e\in\Gamma'_j$, we denote by U_e the group to which e belongs. Additionally, we let $U'_e\subseteq U_e$ be an arbitrary subset of p edges of U_e, including e itself. Notice that $|\Gamma'_j|\ge k^*$ must hold. If $|\Gamma'_j|>k^*$, then we discard edges from Γ'_j arbitrarily, until $|\Gamma'_j|=k^*$ holds. This finishes the description of the grouping. The next theorem establishes some properties of the resulting groupings that will be used later.

THEOREM 3.11.

- *For each $1\le j\le\gamma$, for any pair $X,Y\subseteq\Gamma'_j$ of edge subsets, where $|X|=|Y|$, there is a collection $\mathcal{P}(X,Y):X\overset{1:1}{\rightsquigarrow}_2 Y$ of paths contained in $G[S_j]$, where each path connects a distinct edge of X to a distinct edge of Y, and the paths cause congestion at most 2.*
- *For all $1\le i\ne j\le\gamma$, there is a set $\mathcal{P}_{i,j}:\Gamma'_i\overset{1:1}{\rightsquigarrow}_2\Gamma'_j$ of k^* paths in graph G. That is, each path connects a distinct edge of Γ'_i to a distinct edge of Γ'_j, with total congestion at most 2.*
- *Let $\Gamma^*_1\subseteq\Gamma'_1$ be any subset of k' edges, $\mathcal{M}'\subseteq\mathcal{M}$ any subset of $k'/2$ source-sink pairs, and $\mathcal{T}'$ the subset of terminals participating in pairs in $\mathcal{M}'$. Then there is a set $\mathcal{P}:\mathcal{T}'\overset{1:1}{\rightsquigarrow}_4\Gamma^*_1$ of paths in G, each path connecting a distinct terminal of $\mathcal{T}'$ to a distinct edge of Γ^*_1, with total congestion at most 4.*

Proof. In order to prove the first assertion, fix some $1\le j\le\gamma$. From the integrality of flow, it is enough to prove that there is a flow $F_j(X,Y)$, where each edge in X sends one flow unit, each edge in Y receives one flow unit, the flow congestion is at most 2, and the flow is contained in $G[S_j]$. We start by defining two subsets $X',Y'\subseteq\Gamma_j$ of edges, as follows: $X'=\bigcup_{e\in X}U'_e$ and $Y'=\bigcup_{e\in Y}U'_e$. Observe that $|X'|=|Y'|=|X|\cdot p$. Since set S_j has the $\alpha_{\text{BW}}(k)$-bandwidth property for Γ_j, there

is a flow $F_j(X', Y')$ contained in $G[S_j]$, where every edge in X' sends one flow unit, every edge in Y' receives one flow unit, and the congestion due to this flow is at most $1/\alpha_{\text{BW}}(k)$. We are now ready to define the flow $F_j(X, Y)$. Each edge $e \in X$ spreads one flow unit uniformly among the edges of U'_e along the tree $T_j(U_e)$. Next, all this flow is sent along the flow-paths in $F_j(X', Y')$, where we scale this flow down by factor p. Finally, each edge $e \in Y$ collects all flow from edges in U'_e along the tree $T_j(U_e)$. Since all trees $\{T_U\}_{U \in \mathcal{G}_j}$ are edge-disjoint, and since the congestion caused by $F_j(X', Y')$ is at most $1/\alpha_{\text{BW}}(k) < p$, the resulting flow $F_j(X, Y)$ causes congestion at most 2.

We now turn to prove the second assertion. From the integrality of flow, it is enough to prove that there is a flow $F_{i,j} : \Gamma'_i \rightsquigarrow_2 \Gamma'_j$, where every edge in Γ'_i sends one flow unit and every edge in Γ'_j receives one flow unit. As before, we construct two edge subsets, $X \subseteq \Gamma_j$ and $Y \subseteq \Gamma_i$, as follows: $X = \bigcup_{e \in \Gamma'_i} U'_e$, and $Y = \bigcup_{e \in \Gamma'_j} U'_e$. Notice that $|X| = |Y| = k^* \cdot p$.

Recall that, from the definition of good routers, we already have a flow F_j, where each edge $e \in \Gamma_j$ sends one flow unit to a distinct terminal in $\mathcal{T}$, with total congestion at most $\eta = 2\beta(k)/\alpha_{\text{BW}}(k)$. We discard all flow-paths except those originating at the edges of X. As a result, we obtain a flow F^*_j, where each edge $e \in X$ sends one flow unit to a distinct terminal $t_e \in \mathcal{T}$, and F^*_j causes congestion at most η in G. Let $\mathcal{T}_j$ be the subset of terminals that receive flow in F^*_j, $|\mathcal{T}_j| = |X|$. Similarly, we can define a flow F^*_i, where each edge $e \in Y$ sends one flow unit to a distinct terminal $t_e \in \mathcal{T}$, and F^*_i causes congestion at most η in G. Subset $\mathcal{T}_i$ of terminals is defined similarly. Notice that $\mathcal{T}_i$ and $\mathcal{T}_j$ are not necessarily disjoint. But since the set $\mathcal{T}$ of terminals is $\frac{1}{2}$-well-linked in G, there is a flow $F : \mathcal{T}_i \overset{1:1}{\rightsquigarrow}_2 \mathcal{T}_j$, where each terminal in $\mathcal{T}_i$ sends one flow unit, each terminal in $\mathcal{T}_j$ receives one flow unit, and the total edge-congestion is at most 2. We concatenate the three flows, F^*_i, F, F^*_j, to obtain a flow $F' : X \rightsquigarrow Y$. In this flow, each edge in X sends one flow unit, each edge in Y receives one flow unit, and the total congestion is at most $2\eta + 2$.

We are now ready to define the flow $F_{i,j}$. Each edge $e \in \Gamma'_i$ sends one flow unit along the tree $T_i(U_e)$, which is evenly split among the edges of U'_e. We then use the flow F', scaled down by factor p, to route this flow to the edges of Y. Finally, each edge $e \in \Gamma'_j$ collects the flow that the edges of U'_e receive, along the tree $T_j(U_e)$, so that after collecting all that flow, edge e receives 1 flow unit. In order to analyze the total congestion due to flow $F_{i,j}$, observe that all trees $\{T_i(U)\}_{U \in \mathcal{G}_i} \cup \{T_j(U)\}_{U \in \mathcal{G}_j}$ are edge-disjoint. So the routing along these trees causes a congestion of at most 1. Since flow F' causes congestion of at most $2\eta + 2$ and p is selected so that $p \geq 2\eta + 2$, the congestion due to the scaled-down flow F' is at most 1. The total congestion is therefore at most 2.

Finally, we prove the third assertion. Let $\Gamma^*_1 \subseteq \Gamma'_1$ be any subset of k' edges, $\mathcal{M}' \subseteq \mathcal{M}$ any subset of $k'/2$ source-sink pairs, and $\mathcal{T}'$ the set of all terminals participating in the pairs in $\mathcal{M}'$. Let $X = \bigcup_{e \in \Gamma^*_1} U'_e$, so $|X| = k'p$. As before, we make use of the previously defined flow F_1, where each edge $e \in \Gamma_1$ sends one flow unit to a distinct terminal in $\mathcal{T}$, with total congestion at most $\eta = 2\beta(k)/\alpha_{\text{BW}}(k)$. We discard all flow-paths except those that originate at the edges of X. As a result, we obtain a flow F^*, where each edge $e \in X$ sends one flow unit to a distinct terminal $t_e \in \mathcal{T}$, and F^* causes congestion at most $\eta < p$ in G. We now define a new flow $F^{**} : \Gamma^*_1 \rightsquigarrow_2 \mathcal{T}$, where each edge in Γ^*_1 sends one flow unit, and each terminal in $\mathcal{T}$ receives at most one flow unit. Flow F^{**} is defined as follows. Each edge $e \in \Gamma^*_1$ sends one flow unit to the edges in set U'_e along the tree $T_1(U_e)$, distributing it evenly among these edges.

Each edge in U'_e then sends the $1/p$ flow unit it receives from e to the terminals via the flow F^*, so the flow F^* is scaled down by factor p. Since the congestion caused by flow F^* is $\eta < p$, and the trees $\{T_1(U_e)\}_{e\in\Gamma_1^*}$ are edge-disjoint, the total congestion caused by F^{**} is at most 2. Moreover, each terminal receives at most one flow unit in F^{**}. From the integrality of flow, there is a subset $\mathcal{T}'' \subseteq \mathcal{T}$ of k' terminals, and a collection $\mathcal{P}_1 : \Gamma_1^* \overset{1:1}{\rightsquigarrow}_2 \mathcal{T}''$ of paths in G. Since the set $\mathcal{T}$ of terminals is $\frac{1}{2}$-well-linked, using the integrality of flow, there is a collection $\mathcal{P}_2 : \mathcal{T}'' \overset{1:1}{\rightsquigarrow}_2 \mathcal{T}'$ of paths in G. We then obtain the desired collection $\mathcal{P}$ of paths by concatenating the paths in $\mathcal{P}_1$ with the paths in $\mathcal{P}_2$. □

Step 2: Constructing the trees. The goal of this step is to find a collection $T_1, \ldots, T_{k'}$ of trees in graph G, such that each edge of G belongs to at most 8 trees. For each tree T_i, we will find a subset $E_i \subseteq E(T_i)$ of *special edges*, such that the sets $E_1, \ldots, E_{k'}$ are pairwise disjoint, and for each $1 \le i \le k'$, $E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,\gamma}\}$, where for $1 \le j \le \gamma$, $e_{i,j} \in \Gamma'_j$. Notice that an edge $e \in \Gamma'_j$ may belong to several trees, but only to one of them as a special edge. For each $1 \le j \le \gamma$, we denote $\Gamma_j^* = \{e_{1,j}, \ldots, e_{k',j}\}$ the subset of edges of Γ'_j that the trees $T_1, \ldots, T_{k'}$ contain as special edges. We summarize Step 2 in the next theorem.

THEOREM 3.12. *Given a family $\mathcal{F}$ of good routers and a subset $\Gamma'_j \subseteq \mathrm{out}_G(S_j)$ of edges for each $1 \le j \le \gamma$, as computed in Step 1, we can efficiently find k' trees $T_1, \ldots, T_{k'}$ in graph G, and for each tree T_i a subset $E_i \subseteq E(T_i)$ of special edges, such that*

- *each edge of G belongs to at most 8 trees;*
- *subsets $E_1, \ldots, E_{k'}$ of edges are pairwise disjoint; and*
- *for all $1 \le i \le k'$, $E_i = \{e_{i,1}, \ldots, e_{i,\gamma}\}$, where for all $1 \le j \le \gamma$, $e_{i,j} \in \Gamma'_j$.*

Proof. In order to prove the theorem, we start by augmenting the graph G as follows. First, replace each edge of G with two parallel edges. Next, for each $1 \le j \le \gamma$, add a new vertex s_j, and for each edge $e \in \Gamma'_j$, we subdivide one of the copies of e, by adding a new vertex v_e, which is then connected to the vertex s_j. Notice that from Theorem 3.11, for each $1 \le j \ne j' \le \gamma$, there are exactly k^* edge-disjoint paths connecting s_j to $s_{j'}$ in the resulting graph. Finally, we replace each edge in the resulting graph by two bidirected edges, thus obtaining a directed Eulerian graph that we denote by G^+. From Theorem 3.11, for each pair $1 \le j \ne j' \le \gamma$ of indices, there are k^* edge-disjoint paths connecting s_j to $s_{j'}$, and k^* edge-disjoint paths connecting $s_{j'}$ to s_j. Notice also that each vertex s_j has exactly k^* incoming edges and exactly k^* outgoing edges.

As a next step, we use the standard edge splitting procedure in graph G^+. Our goal is to eventually obtain a graph $\tilde{H}$ over the set $\{s_1, \ldots, s_\gamma\}$ of vertices, such that each pair $s_j, s_{j'}$ is k^*-edge connected, and each edge $e = (s_j, s_{j'}) \in E(\tilde{H})$ is associated with a path P_e connecting s_j to $s_{j'}$ in G^+, while all paths in $\left\{P_e \mid e \in E(\tilde{H})\right\}$ are edge-disjoint in G^+.

Let $D = (V, A)$ be any directed multigraph with no self-loops. For any pair $(v, v') \in V$ of vertices, their connectivity $\lambda(v, v'; D)$ is the maximum number of edge-disjoint paths connecting v to v' in D. Given a pair $a = (u, v)$, $b = (v, w)$ of edges, a splitting-off procedure replaces the two edges a, b by a single edge (u, w). We denote by $D^{a,b}$ the resulting graph. We use the extension of Mader's theorem [47] to directed graphs, due to Frank [29] and Jackson [34]. The following is a simplified version of Theorem 3 from [34].

THEOREM 3.13. *Let $D = (V, A)$ be an Eulerian digraph, $v \in V$, and $a =$*

$(v,u) \in A$. *Then there is an edge* $b = (w,v) \in A$, *such that for all* $y, y' \in V \setminus \{v\}$: $\lambda(y,y';D) = \lambda(y,y';D^{ab})$.

We apply Theorem 3.13 repeatedly to all vertices of G^+ except for the vertices in set $\{s_1,\ldots,s_\gamma\}$, until we obtain a directed graph $\tilde{H}$, whose vertex set is $\{s_1,\ldots,s_\gamma\}$, and for each $1 \le j,j' \le \gamma$, there are k^* edge-disjoint paths connecting s_j to $s_{j'}$ and k^* edge-disjoint paths connecting $s_{j'}$ to s_j. Clearly, each edge $e = (s_j,s_{j'}) \in E(\tilde{H})$ is associated with a path P_e connecting s_j to $s_{j'}$ in G^+, and all paths $\left\{P_e \mid e \in E(\tilde{H})\right\}$ are edge-disjoint. Let $\tilde{H}'$ denote the undirected multigraph identical to $\tilde{H}$, except that now all edges become undirected. Notice that each vertex s_j must have $2k^* \gg \gamma$ edges adjacent to it in $\tilde{H}'$, so the graph contains many parallel edges. For each pair $s_j, s_{j'}$ of vertices, there are exactly $2k^*$ edge-disjoint paths connecting s_j to $s_{j'}$ in $\tilde{H}'$. For convenience, let us denote $2k^*$ by ℓ.

As a next step, we build an auxiliary undirected graph Z on the set $\{s_1,\ldots,s_\gamma\}$ of vertices, as follows. For each pair $s_j, s_{j'}$ of vertices, there is an edge $(s_j,s_{j'})$ in graph Z iff there are at least ℓ/γ^3 edges connecting s_j and $s_{j'}$ in $\tilde{H}'$. If edge $e = (s_j,s_{j'})$ is present in graph Z, then its capacity $c(e)$ is set to be the number of edges connecting s_j to $s_{j'}$ in $\tilde{H}'$. For each vertex s_j, let $C(s_j)$ denote the total capacity of edges incident on s_j in graph Z. We need the following simple observation.

Observation 3.14.

- *For each vertex* $v \in V(Z)$, $(1-1/\gamma^2)\ell \le C(v) \le \ell$.
- *For each pair* (u,v) *of vertices in graph* Z, *we can send at least* $(1-1/\gamma)\ell$ *flow units from* u *to* v *in* Z *without violating the edge capacities.*

Proof. In order to prove the first assertion, recall that each vertex in graph $\tilde{H}'$ has ℓ edges incident to it (this is since, in graph G^+, each vertex $s_1,\ldots,s_\gamma$ had exactly k^* incoming and k^* outgoing edges, and we did not perform edge splitting on these vertices). So $C(v) \le \ell$ for all $v \in V(Z)$. Call a pair $(s_j,s_{j'})$ of vertices bad iff there are fewer than ℓ/γ^3 edges connecting s_j to $s_{j'}$ in $\tilde{H}'$. Notice that each vertex $v \in V(Z)$ may participate in at most γ bad pairs, as $|V(Z)| = \gamma$. Therefore, $C(v) \ge \ell - \gamma\ell/\gamma^3 = \ell(1-1/\gamma^2)$ must hold.

For the second assertion, assume for contradiction that it is not true, and let (u,v) be a violating pair of vertices. Then there is a cut (A,B) in Z, with $u \in A$, $v \in B$, and the total capacity of edges crossing this cut is at most $(1-1/\gamma)\ell$. Since u and v were connected by ℓ edge-disjoint paths in graph $\tilde{H}'$, this means that there are at least ℓ/γ edges in graph $\tilde{H}'$ that connect bad pairs of vertices. But since we can only have at most γ^2 bad pairs, and each pair has fewer than ℓ/γ^3 edges connecting them, this is impossible. □

We now proceed in two steps. First, we show that we can efficiently find a spanning tree of Z with maximum vertex degree at most 3. Next, using this spanning tree, we show how to construct the collection $T_1,\ldots,T_{k'}$ of trees.

Claim 3.15. *There is an efficient algorithm to find a spanning tree* T^* *of* Z *with maximum vertex degree at most* 3.

Proof. We use the algorithm of Singh and Lau [58] for constructing bounded-degree spanning trees. Suppose we are given a graph $G = (V,E)$, and our goal is to construct a spanning tree T of G, where the degree of every vertex is bounded by B. For each subset $S \subseteq V$ of vertices, let $E(S)$ denote the subset of edges with both endpoints in S, and $\delta(S)$ the subset of edges with exactly one endpoint in S. Singh and Lau consider a natural LP relaxation for the problem. We note that their algorithm works for a more general problem where edges are associated with costs,

and the goal is to find a minimum-cost tree that respects the degree requirements; since we do not need to minimize the tree cost, we only discuss the unweighted version here. For each edge $e \in E$, we have a variable x_e indicating whether e is included in the solution. We are looking for a feasible solution to the following LP:

$$\sum_{e \in E} x_e = |V| - 1, \tag{2}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \quad \forall S \subset V, \tag{3}$$

$$\sum_{e \in \delta(v)} x_e \leq B \qquad \forall v \in V, \tag{4}$$

$$x_e \geq 0 \qquad \forall e \in E. \tag{5}$$

Singh and Lau [58] show an efficient algorithm, that, given a feasible solution to the above LP, produces a spanning tree T, where for each vertex $v \in V$, the degree of v is at most $B + 1$ in T. Therefore, in order to prove the claim, it is enough to show a feasible solution to the LP, where $B = 2$. Recall that $|V(Z)| = \gamma$. The solution is defined as follows. Let $e = (u, v)$ be any edge in $E(Z)$. We set the LP-value of e to be $x_e = \frac{\gamma-1}{\gamma} \cdot (\frac{c(e)}{C(v)} + \frac{c(e)}{C(u)})$. We say that $\frac{\gamma-1}{\gamma} \cdot \frac{c(e)}{C(v)}$ is the contribution of v to x_e, and $\frac{\gamma-1}{\gamma} \cdot \frac{c(e)}{C(u)}$ is the contribution of u. We now verify that all constraints of the LP hold.

First, it is easy to see that $\sum_{e \in E} x_e = |V| - 1$, as required. Indeed,

$$\begin{aligned}
\sum_{e \in E} x_e &= \sum_{e=(u,v) \in E} \frac{\gamma - 1}{\gamma} \cdot \left(\frac{c(e)}{C(v)} + \frac{c(e)}{C(u)} \right) \\
&= \sum_{v \in V} \frac{\gamma - 1}{\gamma} \cdot \left(\sum_{e \in \delta(v)} \frac{c(e)}{C(v)} \right) \\
&= \frac{\gamma - 1}{\gamma} \cdot |V| \\
&= |V| - 1,
\end{aligned}$$

as $|V| = \gamma$.

Next, consider some subset $S \subset V$ of vertices. Notice that it is enough to establish constraint (3) for subsets S with $|S| \geq 2$. From Observation 3.14, the total capacity of edges in $E_Z(S, \overline{S})$ must be at least $(1 - 1/\gamma)\ell$. Since for each $v \in S$, $C(v) \leq \ell$, the total contribution of the vertices in S toward the LP-weights of edges in $E_Z(S, \overline{S})$ is at least $\frac{\gamma-1}{\gamma} \cdot (1 - 1/\gamma) = (1 - 1/\gamma)^2$. Therefore,

$$\sum_{e \in E(S)} x_e \leq \frac{\gamma - 1}{\gamma} |S| - (1 - 1/\gamma)^2 = |S| - |S|/\gamma - 1 - 1/\gamma^2 + 2/\gamma \leq |S| - 1$$

since we assume that $|S| \geq 2$. This establishes constraint (3). Finally, we show that for each $v \in V(Z)$, $\sum_{e \in \delta(v)} x_e \leq 2$. First, the contribution of the vertex v to this summation is bounded by 1. Next, recall that for each $u \in V(Z)$, $C(u) \geq (1 - 1/\gamma^2)\ell$, while the total capacity of edges in $\delta(v)$ is at most ℓ. Therefore, the total contribution of other vertices to this summation is bounded by $\frac{\ell}{(1-1/\gamma^2)\ell} \cdot \frac{\gamma-1}{\gamma} \leq \frac{\gamma}{\gamma+1} \leq 1$. The algorithm of Singh and Lau can now be used to obtain a spanning tree T^* for Z with maximum vertex degree at most 3. □

Root the tree T^* at any degree-1 vertex r. Let $e = (s_i, s_j)$ be some edge of the tree, where s_i is the parent of s_j. Recall that there are at least ℓ/γ^3 edges (s_i, s_j) in graph $\tilde{H}'$. Let $A(e)$ be any collection of exactly $\lfloor \ell/\gamma^3 \rfloor$ such edges. Recall that for each edge $\hat{e} \in A(e)$ in graph $\tilde{H}'$, there is a path P, connecting either s_i to s_j or s_j to s_i, in graph G^+ (recall that graph G^+ is directed). Since the direction of the edges in G^+ will not play any role in the following argument, we will assume without loss of generality that P is directed from s_j toward s_i. Recall that the first edge on path P must connect s_j to some vertex $v_{\tilde{e}}$, where $\tilde{e} \in \Gamma'_j$, and similarly, the last edge on path P connects some vertex $v_{\tilde{e}'}$, for $\tilde{e}' \in \Gamma'_i$, to s_i. So by removing the first and the last edges from path P, we obtain a path $P_{\hat{e}}$ in graph G that connects edge $\tilde{e} \in \Gamma'_j$ to edge $\tilde{e}' \in \Gamma'_i$. Since s_i is the parent of s_j in tree T^*, we will think of $P_{\hat{e}}$ as being directed from S_j toward S_i. We call $\tilde{e}$ *the first edge of* $P_{\hat{e}}$ and $\tilde{e}'$ *the last edge of* $P_{\hat{e}}$. Going back to the edge $e = (s_i, s_j)$ in tree T^*, we can now define a set $\mathcal{P}(e) = \{P_{\hat{e}} \mid \hat{e} \in A(e)\}$ of exactly $\lfloor \ell/\gamma^3 \rfloor$ paths in graph G, associated with e. We let

$$B_1(e) = \left\{\tilde{e} \in \Gamma'_j \mid \tilde{e} \text{ is the first edge on some path } P_{\hat{e}} \in \mathcal{P}(e)\right\}$$

and

$$B_2(e) = \left\{\tilde{e} \in \Gamma'_i \mid \tilde{e} \text{ is the last edge on some path } P_{\hat{e}} \in \mathcal{P}(e)\right\}.$$

Both sets $B_1(e)$, $B_2(e)$ are multisets; that is, if some edge $\tilde{e} \in \Gamma'_j$ appears as a first edge on two paths in $\mathcal{P}(e)$, then we add two copies of $\tilde{e}$ to $B_1(e)$. (From the construction of G^+, it is easy to see that $\tilde{e}$ may appear as the first edge on at most two such paths.) We then have that $\mathcal{P}(e) : B_1(e) \overset{1:1}{\rightsquigarrow}_4 B_2(e)$ in graph G, since, from the construction of graphs G^+ and $\tilde{H}'$, every edge of graph G may appear on at most four paths of $\bigcup_{e \in E(T^*)} \mathcal{P}(e)$.

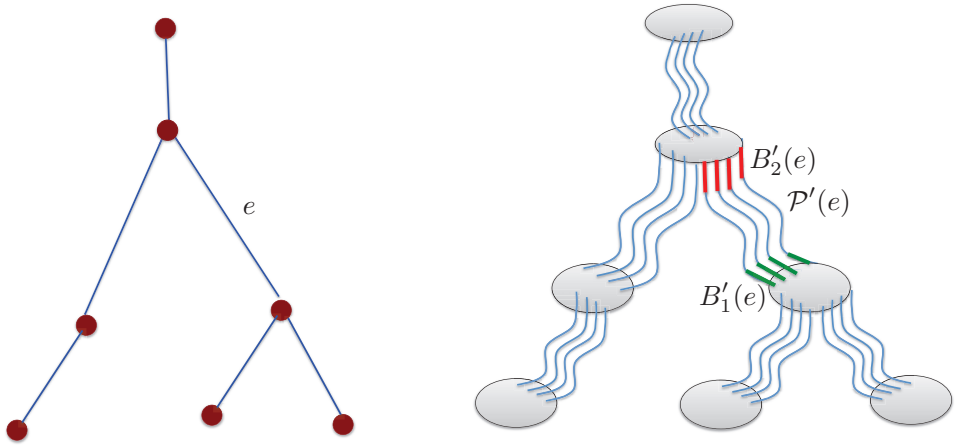
We call the sets $B_1(e), B_2(e)$ of edges *bundles corresponding to* e, and we view $B_1(e)$ as a bundle that belongs to S_j, while $B_2(e)$ is a bundle that belongs to S_i. Since the degree of tree T^* is at most 3, every set S_j has at most three bundles that belong to it. From the construction of graph G^+, for every vertex $s_i : 1 \leq i \leq \gamma$, each edge in Γ'_i may appear at most twice in the multiset defined by the union of the bundles that belong to S_i. In particular, it is possible that it appears twice in the same bundle. We need to make sure that this never happens. In order to achieve this, we will define, for each edge $e \in E(T^*)$, smaller bundles, $B'_1(e) \subseteq B_1(e)$ and $B'_2(e) \subseteq B_2(e)$, such that each edge appears at most once in each bundle, and there is a subset $\mathcal{P}'(e) \subseteq \mathcal{P}(e)$, where $\mathcal{P}'(e) : B'_1(e) \overset{1:1}{\rightsquigarrow}_4 B'_2(e)$. We will also ensure that $|B'_1(e)| = |B'_2(e)| = \lfloor \frac{\ell}{4\gamma^3} \rfloor$ (see Figure 2).

This is done as follows. Consider some edge $e = (s_i, s_j)$ in tree T^*, and assume that s_i is the parent of s_j in the tree. Consider first $B_1(e)$. For each edge $\tilde{e} \in B_1(e)$, if two copies of $\tilde{e}$ appear in $B_1(e)$, then we remove one of the copies from $B_1(e)$. If $P \in \mathcal{P}(e)$ is one of the two paths for which $\tilde{e}$ is the first edge, then we remove P from $\mathcal{P}(e)$, and we also remove its last edge from $B_2(e)$. It is easy to see that we remove at most half the edges of $B_1(e)$. We then perform the same operation for $B_2(e)$. In the end, both $B_1(e)$ and $B_2(e)$ must contain at least a $1/4$ of the original edges, and $\mathcal{P}(e)$ contains at least a $1/4$ of the original paths. We now let $\mathcal{P}'(e)$ be any subset of exactly $\lfloor \frac{\ell}{4\gamma^3} \rfloor$ remaining paths, and we set $B'_1(e)$ to be the set of all edges $\tilde{e}$ that appear as the first edge on some path in $\mathcal{P}'(e)$, and, similarly, we set $B'_2(e)$ to be the set of all edges that appear as the last edge on some path in $\mathcal{P}'(e)$. We perform this operation for all edges e of tree T^*. We then obtain the following corollary.

COROLLARY 3.16. *Our algorithm finds a tree T^* with maximum vertex degree at*

most 3, *rooted at a degree-*1 *vertex, with vertex set* $V(T^*) = \{v_1, \ldots, v_\gamma\}$. *Additionally, for each edge* $e = (v_i, v_j) \in E(T^*)$, *where* v_i *is a parent of* v_j *in* T^*, *the algorithm finds a collection* $\mathcal{P}'(e)$ *of paths in graph* G, *a subset* $B_1'(e) \subseteq \Gamma(S_j)$ *of* $\left\lfloor \frac{\ell}{4\gamma^3} \right\rfloor$ *edges, and a subset* $B_2'(e) \subseteq \Gamma(S_i)$ *of* $\left\lfloor \frac{\ell}{4\gamma^3} \right\rfloor$ *edges, such that*

- *each path in* $\mathcal{P}'(e)$ *connects a distinct edge of* $B_1'(e)$ *to a distinct edge of* $B_2'(e)$ *in* G; *and*
- *the paths in* $\bigcup_{e \in E(T^*)} \mathcal{P}'(e)$ *cause total edge-congestion at most* 4 *in* G.



(a) Tree T^*

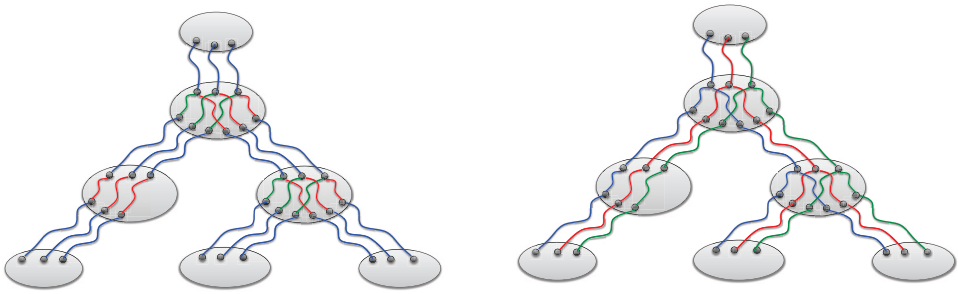
(b) Sets $\mathcal{P}'(e)$ of paths; the edges of $B_1'(e)$ are shown in green, and the edges of $B_2'(e)$ in red.

FIG. 2. *Illustration for Corollary* 3.16.

We are now ready to define the subsets $\Gamma_j^* \subseteq \Gamma_j'$ of k' edges, $\Gamma_j^* = \{e_{1,j}, \ldots, e_{k',j}\}$, that our trees will span. Fix some index $1 \leq j \leq \gamma$. If s_j is not the root of the tree T^*, then we let $\Gamma_j^* = B_1(e)$, where e is the edge connecting s_j to its parent in T^*. If s_j is the root of the tree, then $\Gamma_j^* = B_2(e)$, where e is the unique edge incident on s_j in tree T^*. Notice that $|\Gamma_j^*| = \lfloor \frac{\ell}{4\gamma^3} \rfloor = \lfloor \frac{k^*}{2\gamma^3} \rfloor = k'$.

Finally, we construct the trees $T_1, \ldots, T_{k'}$. In order to construct these trees, for each vertex $s_j \in V(T^*)$, we define a set $\mathcal{Q}_j$ of paths, and a corresponding set $U_j \subseteq S_j$ of special vertices in graph G, as follows. If the degree of s_j is 1 in T^*, then let e be the unique edge of T^* incident on s_j. We let U_j be the set of vertices of S_j that serve as endpoints of the paths in $\mathcal{P}'(e)$, and we set $\mathcal{Q}_j = \emptyset$. If the degree of s_j is 2 in T^*, then let e_1 and e_2 be the two edges of T^* incident on s_j, and assume that e_1 is the edge connecting s_j to its parent. Let U^1 be the set of vertices of S_j that serve as endpoints of the paths in $\mathcal{P}'(e_1)$, and let U^2 be defined similarly for e_2. We then define $U_j = U^1 \cup U^2$. Notice that from Theorem 3.11, we can find a set $\mathcal{Q}_j : U^1 \overset{1:1}{\rightsquigarrow}_2 U^2$ of paths contained in $G[S_j]$, where each path in $\mathcal{Q}_j$ connects a distinct vertex of U' to a distinct vertex of U''. Finally, assume that s_j has degree 3 in T^*. Let e_1 be the edge incident on s_j that connects it to its parent, and let e_2, e_3 be the other two edges incident on s_j in T^*. Let U^1 and U^2 be defined exactly as before, with respect to e_1 and e_2. Similarly, we let U^3 be the set of vertices of S_j which serve as endpoints of the paths in $\mathcal{P}'(e_3)$. We let $U_j = U^1 \cup U^2 \cup U^3$. As before, from Theorem 3.11, we can find a set $\mathcal{Q}^1 : U^1 \overset{1:1}{\rightsquigarrow}_2 U^2$ and a set $\mathcal{Q}^2 : U^1 \overset{1:1}{\rightsquigarrow}_2 U^3$ of paths in $G[S_j]$. We let

$\mathcal{Q}_j = \mathcal{Q}^1 \cup \mathcal{Q}^2$ (see Figure 3). Let $\mathcal{P} = \bigcup_{e \in E(T^*)} \mathcal{P}'(e)$ and $\mathcal{Q} = \bigcup_{v_j \in V(T^*)} \mathcal{Q}_j$. Recall that every edge of G belongs to at most four paths in $\mathcal{P}$, and it is easy to see that it belongs to at most four paths in $\mathcal{Q}$.



(a) Paths in $\mathcal{P}$ are shown in blue, and paths in $\mathcal{Q}$ are shown in red and green.

(b) The final construction of the trees.

FIG. 3. *Constructing the trees.*

It is now easy to complete the construction of the trees. Let $\mathcal{U}$ be the graph obtained from disjoint union of the paths in $\mathcal{P}$ and $\mathcal{Q}$; that is, if a vertex, or an edge, belongs to several paths in $\mathcal{P} \cup \mathcal{Q}$, then we create several copies of this edge or vertex. For each $s_j \in V(T^*)$, for each vertex $v \in U_j$, we unify all copies of v that serve as endpoints of the paths in $\mathcal{P} \cup \mathcal{Q}$ (there can be at most three such copies), and we call the resulting copy of v a *special copy*. (Notice that v may belong to additional paths in $\mathcal{P} \cup \mathcal{Q}$ as an inner vertex, and we do not unify such copies of v.) It is easy to see that the resulting graph is a collection of k' disjoint trees $T_1, \ldots, T_{k'}$, where for each edge $e \in E(T^*)$, each path $P \in \mathcal{P}'(e)$ is contained in a distinct tree T_i (see Figure 3). Consider now some vertex $v_j \in V(T^*)$ and the corresponding set Γ^*_j of k' edges. If s_j is not the root of T^*, then let e be the edge of T^* connecting s_j to its parent; otherwise, let e be the unique edge of T^* incident on s_j. For each edge $\tilde{e} \in \Gamma^*_j$, the unique path $P \in \mathcal{P}'(e)$ that originates from $\tilde{e}$ is contained in some tree T_i for $1 \le i \le k'$, and we add $\tilde{e}$ to the set E_i of special edges for T_i. It is easy to see that each edge in Γ^*_j is added as a special edge to a distinct tree in $\{T_1, \ldots, T_{k'}\}$. As observed before, every edge of G may belong to at most eight trees T_i for $1 \le i \le k'$. Notice that for $1 \le i \le k'$, tree T_i defines a connected subgraph of G that may not necessarily be a tree (since the same edge or vertex may appear twice in T_i in graph $\mathcal{U}$). We simply delete edges from the resulting subgraph of G until it becomes a tree, and we still denote that tree by T_i. □

Step 3: Constructing the expander and finding the routing. In this step, we construct the expander X, together with its embedding into the graph G, and find the final routing of a subset of demands in $\mathcal{M}$. Let $\mathcal{M}' \subseteq \mathcal{M}$ be any subset of $k'/2$ demand pairs, and let $\mathcal{T}'$ be the subset of terminals participating in the pairs of $\mathcal{M}'$.

Let $\mathcal{P}^* = \mathcal{T}' \overset{1:1}{\rightsquigarrow}_4 \Gamma^*_1$ be the collection of paths connecting the terminals of $\mathcal{T}'$ to the edges of $\Gamma^*_1 \subseteq \Gamma_1$ (where $\Gamma^*_1 = \{e_{1,1}, \ldots, e_{k',1}\}$), guaranteed by Theorem 3.11. Denote $\mathcal{P}^* = \{P_t \mid t \in \mathcal{T}'\}$, where P_t is the path originating from terminal t. Rename the terminals in $\mathcal{T}'$ as $\mathcal{T}' = \{t_1, \ldots, t_{k'}\}$, where for each $1 \le i \le k'$, t_i is the terminal whose path P_{t_i} terminates at the edge $e_{i,1}$ (the unique edge of Γ^*_1 that belongs to the tree T_i as a special edge). For $1 \le i \le k'$, let C_i be the connected subgraph of graph G that consists of the union of the tree T_i and the path P_{t_i}. Since each edge of graph

G participates in at most eight trees T_i, and at most four paths in $\mathcal{P}^*$, each edge of G participates in at most 12 connected subgraphs C_i.

We now construct the expander X and embed it into the graph G. The set of vertices of X is $V(X) = \{v_1, \ldots, v_{k'}\}$, where we view each vertex v_i as representing the terminal $t_i \in \mathcal{T}'$. We view the connected subgraph C_i of G as the embedding of the vertex v_i into G. Finally, we need to define the set of the edges of X and specify their embedding into G. In order to do so, we use the cut-matching game from Theorem 2.2. Recall that in each iteration j, the cut player produces a partition (A_j, B_j) of $V(X)$, with $|A_j| = |B_j|$. The matching player then returns some matching M_j between the vertices of A_j and B_j, and the edges of M_j are added to graph X. We use the graphs $G[S_j]$ to route the matchings M_j. Specifically, let (A_1, B_1) be the partition of $V(X)$ produced by the cut player in the first iteration. Consider the set $\Gamma_1^* = \{e_{1,1}, \ldots, e_{k',1}\}$ of edges. Partition (A_1, B_1) of $V(X)$ defines a partition (A_1', B_1') of these edges, where $A_1' = \{e_{i,1} \mid v_i \in A_1\}$ and $B_1' = \{e_{i,1} \mid v_i \in B_1\}$. From Theorem 3.11, we can find a set $\mathcal{R}_1 : A_1' \overset{1:1}{\rightsquigarrow}_2 B_1'$ of $|A_1'|$ paths contained in $G[S_1]$, where each path in $\mathcal{R}_1$ connects a distinct edge of A_1' to a distinct edge of B_1'. Set $\mathcal{R}_1$ of paths then defines a matching M_1' between the sets A_1' and B_1', which in turn defines a matching M_1 between the sets A_1 and B_1 of vertices of $V(X)$. We then treat M_1 as the response of the matching player. For each edge $e = (v_i, v_{i'}) \in M_1$ of the matching, we let P_e be the unique path of $\mathcal{R}_1$ connecting $e_{i,1}$ to $e_{i',1}$. We view P_e as the embedding of e into graph G. We continue similarly to execute the remaining iterations, where in each iteration $j : 1 \le j \le \gamma$, we use the set $S_j \in \mathcal{F}$ to find the matching M_j. That is, we define the partition (A_j', B_j') of Γ_j' based on the partition (A_j, B_j) of $V(X)$ as before, and we find a collection $\mathcal{R}_j : A_j' \overset{1:1}{\rightsquigarrow}_2 B_j'$ of paths contained in $G[S_j]$. These paths give us the matching M_j' between the sets A_j' and B_j' of edges, which in turn gives us the matching M_j between the sets A_j and B_j of vertices of $V(X)$. For each edge $e = (v_i, v_{i'}) \in M_j$, we let P_e be the unique path of $\mathcal{R}_j$ connecting $e_{i,j}$ to $e_{i',j}$. We view P_e as the embedding of e into graph G. The final graph X is the graph obtained after γ iterations, with $E(X) = \bigcup_{j=1}^{\gamma} M_j$, and we are guaranteed that with constant probability it is an $\Omega(\log k')$-expander. For each edge $e = (v_i, v_{i'}) \in E(X)$, we have defined an embedding P_e of e into G, where P_e is a path connecting some vertex in C_i to some vertex in $C_{i'}$. Let $\mathcal{P}_X = \{P_e \mid e \in E(X)\}$. Then $\mathcal{P}_X = \bigcup_{j=1}^{\gamma} \mathcal{R}_j$, and the total congestion caused by paths in $\mathcal{P}_X$ in G is at most 2. This finishes the definition of the expander X and of its embedding into G.

We now use the expander X and its embedding into G to route a subset of demand pairs. We identify from now on the vertices of X with the terminals of $\mathcal{T}'$ they represent; that is, $V(X) = \mathcal{T}'$.

We use Theorem 2.1 to find a collection $\tilde{\mathcal{P}}$ of $r = \Omega(\frac{k'}{\gamma^2})$ vertex-disjoint paths in the expander X, routing r distinct demand pairs. Let $\mathcal{M}'' \subseteq \mathcal{M}'$ be the set of these demand pairs, and assume, by renumbering the terminals as necessary, that $\mathcal{M}'' = \{(t_1, t_2), (t_3, t_4), \ldots, (t_{2r-1}, t_{2r})\}$. For each $1 \le i \le r$, let $P_i \in \tilde{\mathcal{P}}$ be the path connecting t_{2i-1} to t_{2i}. In order to complete the routing, we transform each such path P_i into a path Q_i in graph G, connecting the same pair (t_{2i-1}, t_{2i}) of terminals.

Fix some $1 \le i \le r$. We now show how to transform the path P_i connecting t_{2i-1} to t_{2i} in graph X to a path Q_i connecting the same pair of terminals in graph G. In order to do so, we will replace the edges and the vertices of path P_i by paths in graph G. We will think of P_i as directed from t_{2i-1} to t_{2i}. First, each edge $e = (t_a, t_b) \in P_i$ is replaced by the path $P_e \subseteq G$, connecting some vertex $v \in C_a$ to some vertex $u \in C_b$ (where P_e is the path into which e is embedded). Next, consider some inner vertex

$t_x \in P_i$, and let e, e' be the two edges appearing immediately before and immediately after t_x on the original path P_i, respectively. Let $v_x \in C_x$ be the last vertex on path P_e, and let $v'_x \in C_x$ be the first vertex on path $P_{e'}$. Then we replace the vertex t_x with an arbitrary path P_x connecting v_x to v'_x in the connected subgraph C_x of G. It now only remains to take care of the endpoints of path P_i. Let e be the first edge on the original path P_i, and recall that the first vertex on P_i is t_{2i-1}. Let $v_{2i-1} \in C_{2i-1}$ be the first vertex on the path P_e. Then we replace t_{2i-1} by any path connecting t_{2i-1} to v_{2i-1} in the connected subgraph C_{2i-1} of G. The last vertex of P_i is taken care of similarly. Let Q_i denote the resulting path. Notice that Q_i consists of two types of segments: the first type is the paths P_e for edges $e \in P_i$, and the second type is the paths P_x for vertices $x \in P_i$. Let $Q_1, \ldots, Q_r$ be the resulting set of paths. We now bound the congestion due to paths in $Q_1, \ldots, Q_r$ in graph G. Recall that the paths $\{P_i\}_{i=1}^r$ are edge- and vertex-disjoint. Recall also that each edge of graph G participates in at most two paths of the set $\mathcal{P}_X = \{P_e \mid e \in E(X)\}$. Therefore, the congestion due to type-1 segments in $\{Q_i\}_{i=1}^r$ is at most 2. Since the paths in $\{P_i\}_{i=1}^r$ are vertex-disjoint, and every edge of graph G participates in at most 12 components $C_1, \ldots, C_{k'}$, the congestion due to type-2 segments is bounded by 12. Overall, the paths in $\{Q_i\}_{i=1}^r$ cause congestion at most 14. The number of demand pairs routed is $r = \Omega(\frac{k'}{\gamma^2}) = \Omega(\frac{k}{\log^{20.5} k \log\log k})$.

To conclude, we have started with a graph G, a collection $\mathcal{M}$ of k source-sink pairs, and the set $\mathcal{T}$ of terminals participating in pairs in $\mathcal{M}$, such that $\mathcal{T}$ is $\frac{1}{2}$-well-linked for G. We have constructed a routing for the subset $\mathcal{M}'' \subseteq \mathcal{M}$ of $\Omega(\frac{k}{\log^{20.5} k \log\log k})$ pairs with congestion at most 14.

We note that our algorithm uses randomization in two places: first, in constructing a random partition of vertices of the legal contracted graph into clusters in Claim 3.6, and second, in the cut-matching game, in order to embed an expander into our graph. As observed before, the random partitioning of the legal contracted graph can be run $\mathrm{poly}(n)$ times to ensure that the procedure succeeds with high probability. The cut-matching game only returns an $\Omega(\log k')$-expander with constant probability. If the graph returned by the cut-matching game is not an expander, then we may fail to route the desired number of demand pairs on node-disjoint paths in the expander. In this case, we can terminate the algorithm execution, and repeat it, starting from the step where the cut-matching game is executed. Clearly, after $\mathrm{poly}(n)$ iterations, we are guaranteed that the algorithm will succeed with high probability.

Since we lose an additional $O(\log^2 k)$ factor on the number of pairs routed due to the preprocessing step that ensures $\frac{1}{2}$-well-linkedness of the terminals, our algorithm routes $\Omega(\frac{\mathsf{OPT}}{\log^{22.5} k \log\log k})$ pairs with congestion at most 14 with high probability.

4. Routing with grouping. The goal of this section is to prove Theorem 1.2. We roughly follow the algorithm from section 3, except that we use a slightly different theorem for routing on expanders, summarized below. Its proof is similar to some arguments from [10], and it uses the new constructive proof of the Lovasz local lemma by Moser and Tardos [50]. The proof is deferred to Appendix C.

THEOREM 4.1. *Let $G = (V, E)$ be any n-vertex α-expander (for $\alpha \leq 1$) with maximum degree $d_{\max}$, and let $c \geq 1$ be any integer. Then there is a value $m = \Theta(d_{\max}^{1+3/c}(\log n)^{1+5/c}/\alpha^{1+3/c})$, such that, for any partition $\mathcal{G} = (V_1, \ldots, V_r)$ of the vertices of G into groups of size at least m, and for any partial matching $M \subseteq ([r] \times [r])$, we can efficiently find, for each pair $(i, j) \in M$, a path $P_{i,j}$ connecting a vertex of V_i to a vertex of V_j, such that with high probability, the set of paths $\{P_{i,j} \mid (i, j) \in M\}$*

causes vertex congestion *at most c in G.*

Assume that we are given a graph $G = (V, E)$ and a set $\mathcal{T} \subseteq V$ of k_0 terminals, such that $\mathcal{T}$ is α_0-well-linked in G. We will construct an expander X on a subset of terminals in $\mathcal{T}$ as in section 3, where X is a $\frac{1}{2}$-expander, $|V(X)| \leq k_0$, and the maximum degree of X is bounded by $\gamma_{\text{CMG}}(k_0) = O(\log^2 k_0)$. We denote by $m = \Theta\left((\log k_0)^{3+11/c}\right)$ the corresponding parameter from Theorem 2.1 for this setting.

We also use the following parameters. Let $\tilde{k} = \frac{k_0\alpha_0}{12}$. Recall that we have defined, in section 3, a parameter k', whose value is $\Omega(\frac{k}{\log^{16.5} k \log\log k})$, where k is the number of the terminals. We define a function $q(k) = O(\log^{16.5} k \log\log k)$, so that for any integer k, $k' = k/q(k)$. We then set $\tilde{k}' = \tilde{k}/q(\tilde{k}) = \Omega(\frac{k_0\alpha_0}{\log^{16.5}\tilde{k}\log\log\tilde{k}}) = \Omega(\frac{k_0\alpha_0}{\log^{17} k_0})$. Intuitively, we will define a grouping $\mathcal{G}''$ of the terminals in $\mathcal{T}$ into groups of size roughly $k_0\alpha_0$ and select one representative terminal from each group. Let $\mathcal{T}''$ denote the resulting set of terminals. We will show that the set $\mathcal{T}''$ is $\frac{1}{2}$-well-linked in graph G, and $|\mathcal{T}''| \geq \tilde{k}$. We can then apply the algorithm from section 3 to construct an expander X over a subset $\mathcal{T}' \subseteq \mathcal{T}''$ of $\tilde{k}'$ terminals. These terminals are in turn grouped into groups of size at least m, and we then apply Theorem 2.1 to route these terminals in the expander X. We now proceed with a formal description of the algorithm.

We define three hierarchical groupings of the terminals in $\mathcal{T}$. Let T be any spanning tree of the graph G. Our first step is to group the terminals in $\mathcal{T}$ into groups of size roughly $k_0 m/\tilde{k}'$. To do so, we use the grouping technique with the parameter $6\lceil k_0 m/\tilde{k}'\rceil$ on the set $\mathcal{T}$ of terminals and the tree T. As a result, we obtain a partition $\mathcal{G}$ of the set $\mathcal{T}$ of terminals into groups of size at least $6\lceil k_0 m/\tilde{k}'\rceil$ and at most $18\lceil k_0 m/\tilde{k}'\rceil$. For each group $U \in \mathcal{G}$, there is a tree T_U spanning the terminals of U, and all the trees in $\{T_U\}_{U\in\mathcal{G}}$ are edge-disjoint. The final grouping of the terminals returned by the algorithm is $\mathcal{G}$. The size of each group in $\mathcal{G}$ is bounded by $18\lceil k_0 m/\tilde{k}'\rceil = O(k_0(\log k_0)^{3+11/c} \cdot \frac{\log^{17} k_0}{k_0\alpha_0}) = O(\frac{(\log k_0)^{21+11/c}}{\alpha_0})$, as required. Assume now that we are given a set $\mathcal{M}$ of integral $(1, \mathcal{G})$-restricted demands on $\mathcal{T}$. We now show an algorithm to integrally route the demands in $\mathcal{M}$.

For each group $U \in \mathcal{G}$, we further partition the terminals in U into at least m groups of roughly equal size, using the tree T_U. Let $n_U = |U|$. We use the grouping technique with the parameter $\frac{1}{3}\lfloor n_U/m\rfloor$ for U and tree T_U. We then obtain at least m groups, whose sizes are at least $\frac{k_0}{\tilde{k}'}$ and at most $\frac{36k_0}{\tilde{k}'}$. For each group $U \in \mathcal{G}$, let $\mathcal{P}(U)$ denote the resulting partition of U, and let $\mathcal{G}' = \bigcup_{U\in\mathcal{G}} \mathcal{P}(U)$ be the corresponding grouping of the terminals. Notice that, again, for each group $U' \in \mathcal{G}'$, we have a tree $T_{U'}$ spanning the terminals of U', such that all trees in $\{T_{U'}\}_{U'\in\mathcal{G}'}$ are edge-disjoint.

Finally, for each group $U' \in \mathcal{G}'$, we further partition the terminals in U' into groups of size at least $\lceil\frac{1}{\alpha_0}\rceil$ and at most $3\lceil\frac{1}{\alpha_0}\rceil$, using the standard grouping technique on the tree $T_{U'}$, with the parameter $\lceil\frac{1}{\alpha_0}\rceil$. For each set $U' \in \mathcal{G}'$, let $\mathcal{P}'(U')$ be the resulting partition of U', and let $\mathcal{G}'' = \bigcup_{U'\in\mathcal{G}'} \mathcal{P}'(U')$ be the resulting partition of the terminals. For each set $U'' \in \mathcal{G}''$, let $t_{U''}$ be any representative terminal from U'', and let $\mathcal{T}'' = \{t_{U''} \mid U'' \in \mathcal{G}''\}$. Each group $U'' \in \mathcal{G}''$ is again associated with a tree $T_{U''}$ spanning the terminals of U'', and all trees in $\{T_{U''}\}_{U''\in\mathcal{G}''}$ are edge-disjoint. We start with the following simple claim.

CLAIM 4.2. *The terminals in $\mathcal{T}''$ are $\frac{1}{2}$-well-linked.*

Proof. Let $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{T}''$ be any pair of equal-sized disjoint subsets of terminals. From Observation 2.4, it is enough to show that there is a flow $F : \mathcal{T}_1 \overset{1:1}{\rightsquigarrow}_2 \mathcal{T}_2$ in G. We construct two sets X, Y of terminals as follows. For each terminal $t \in \mathcal{T}_1 \cup \mathcal{T}_2$, let

S_t be any subset of $\lceil 1/\alpha_0 \rceil$ vertices of the set $U'' \in \mathcal{G}''$, where $t \in U''$. We then let $X = \bigcup_{t \in \mathcal{T}_1} S_t$, and $Y = \bigcup_{t \in \mathcal{T}_2} S_t$. Since the terminals in $\mathcal{T}$ are α_0-well-linked in G, there is a flow $F' : X \overset{1:1}{\rightsquigarrow}_{1/\alpha} Y$ in G. Scaling this flow down by factor $\lceil 1/\alpha \rceil$, we obtain a flow F'' where every vertex of X sends $\frac{1}{\lceil 1/\alpha \rceil}$ flow units, every vertex of Y receives $\frac{1}{\lceil 1/\alpha \rceil}$ flow units, and the total edge-congestion is at most 1. In order to obtain the final flow F, every terminal $t \in \mathcal{T}_1$ spreads 1 flow unit evenly among the vertices of S_t, along the corresponding tree $T_{U''}$, where $t \in U''$. Every vertex of X then receives $\frac{1}{\lceil 1/\alpha \rceil}$ flow units. We then use flow F'' to ensure that every vertex of Y receives $\frac{1}{\lceil 1/\alpha \rceil}$ flow units. Finally, every terminal $t \in \mathcal{T}_2$ collects one flow unit from the vertices of S_t along the corresponding tree $T_{U''}$, where $t \in U''$. It is easy to see that the congestion due to flow F is at most 2. □

Notice that the number of terminals in $\mathcal{T}''$ is $|\mathcal{T}''| \geq \frac{k_0}{3\lceil 1/\alpha_0 \rceil} \geq \frac{k_0 \alpha_0}{12} \geq \tilde{k}$. Therefore, we now have a graph G and a subset $\mathcal{T}''$ of at least $\tilde{k}$ terminals that are $\frac{1}{2}$-well-linked in G. As before, we perform the standard transformation that ensures that the degree of every vertex is at most 4, and the degree of every terminal in $\mathcal{T}''$ is 1. It is easy to verify that the terminals of $\mathcal{T}''$ remain $\frac{1}{2}$-well-linked in the resulting graph G', since for any pair $\mathcal{T}_1, \mathcal{T}_2$ of equal-sized subsets of $\mathcal{T}''$, there is a flow $F : \mathcal{T}_1 \overset{1:1}{\rightsquigarrow}_2 \mathcal{T}_2$ in G iff it exists in G'. Therefore, we obtain precisely the starting point of the algorithm in section 3. We can now use the algorithm from section 3 to construct the expander X on a subset $\mathcal{T}' \subseteq \mathcal{T}''$ of $\tilde{k}'$ terminals. The only difference is that, instead of selecting an arbitrary subset $\mathcal{T}'$ of terminals as in the algorithm, we select $\mathcal{T}'$ as follows. Consider the grouping $\mathcal{G}'$ of the terminals. Let $\mathcal{G}^*$ be the grouping of the terminals in $\mathcal{T}''$ that $\mathcal{G}'$ induces. We select one representative terminal from each group in $\mathcal{G}^*$, and we let $\mathcal{T}'$ be the set of all selected terminals. By our construction, $|\mathcal{T}'| = |\mathcal{G}'| \leq 6m|\mathcal{G}| \leq \frac{6k_0 m}{6k_0 m/\tilde{k}'} \leq \tilde{k}'$.

We now use the algorithm from section 3 to construct a $\frac{1}{2}$-expander X on the set $\mathcal{T}'$ of terminals and embed it into G. Recall that for each terminal $t \in \mathcal{T}'$, we have a connected subgraph C_t of G, and each edge of G participates in at most 12 such subgraphs. Each edge e of X is mapped to a path P_e in G, and each edge of G participates in at most two such paths.

Consider the grouping $\mathcal{G}^{**}$ of the terminals in $\mathcal{T}'$ induced by $\mathcal{G}$. In order to obtain $\mathcal{G}^{**}$, we start from $\mathcal{G}$, and we ignore terminals that do not belong to $\mathcal{T}'$. By our construction, each group in $\mathcal{G}^{**}$ contains at least m terminals from $\mathcal{T}'$. We assume without loss of generality that the input set of demands is $\mathcal{M} = \{(t_1, t_2), \ldots, (t_{2r-1}, t_{2r})\}$, and for each $1 \leq i \leq 2r$, $t_i \in U_i$, where $U_i \in \mathcal{G}$. Let $U_i^* = U_i \cap \mathcal{T}'$, and recall that $|U_i^*| \geq m$, and $U_i^* \in \mathcal{G}^{**}$. We now use Theorem 2.1 on graph X, set $\mathcal{T}'$ of terminals, grouping $\mathcal{G}^{**}$, and matching $M' = \{(1,2), (3,4), \ldots, (2r-1, 2r)\}$. Let $\mathcal{P}''$ be the set of paths returned by the theorem, where for each $1 \leq j \leq r$, there is a path $P_j'' \in \mathcal{P}$ connecting some terminal $t'_{2j-1} \in U_{2j-1}$ to some terminal $t'_{2j} \in U_{2j}$. The paths in $\mathcal{P}''$ cause vertex congestion at most c in X. We transform these paths into a set $\mathcal{P}'$ of paths connecting the same pairs of terminals in graph G. Since each edge of G participates in at most 12 subgraphs C_i, and at most two paths in set $\{P_e \mid e \in E(X)\}$, the total congestion caused by paths in $\mathcal{P}'$ is at most $14c$. For each $1 \leq j \leq r$, let $P_j' \in \mathcal{P}'$ be the path connecting t'_{2j-1} to t'_{2j}.

We then construct a path P_j connecting t_{2j-1} to t_{2j} as follows: first connect t_{2j-1} to t'_{2j-1} via the tree $T_{U_{2j-1}}$, then use the path P_j' to connect t'_{2j-1} to t'_{2j}, and finally connect t'_{2j} to t_{2j} via the tree $T_{U_{2j}}$. Let $\mathcal{P} = \{P_j\}_{j=1}^r$ be the final routing. Then the total edge congestion caused by $\mathcal{P}$ is bounded by $14c + 1$.

5. Integral sparsifiers. In this section we prove Theorem 1.3. Notice that we can assume without loss of generality that the degree of every terminal is 1, and the number of terminals is d: for each terminal $t \in \mathcal{T}$, we can simply subdivide every edge e incident on t with a new vertex v_t, let S_t be the set of these new vertices, and set $\mathcal{T}' = \bigcup_{t \in \mathcal{T}} S_t$. Let G' be the subgraph of the resulting graph induced by $(V \setminus \mathcal{T}) \cup \mathcal{T}'$, and let $\mathcal{T}'$ be the new set of terminals. Then every terminal in $\mathcal{T}'$ has degree 1, and $|\mathcal{T}'| = d$. Moreover, if H' is a quality (q_1, q_2) integral sparsifier for G', we can obtain a sparsifier H for G by unifying, for each $t \in \mathcal{T}$, all vertices in S_t into a single vertex t in graph H'. It is immediate to verify that the resulting graph H is a quality (q_1, q_2)-sparsifier for G.

For convenience, from now on we assume that every terminal in $\mathcal{T}$ has degree 1, and we denote by k the number of terminals in $\mathcal{T}$. We now show a construction of a sparsifier for G of size $O(k)$.

We first consider a special case where the set $\mathcal{T}$ of terminals is $\alpha_{\text{BW}}(k)$-well-linked in graph G. We use Theorem 1.2 with $c = 1$ and $\alpha_0 = \alpha_{\text{BW}}(k)$ to find a partition $\mathcal{G}$ of the terminals into subsets of size $z = O(\log^{32} k/\alpha_{\text{BW}}(k)) = O(\log^{35.5} k)$. Recall that in the proof of Theorem 1.2, we have constructed, for each group $U \in \mathcal{G}$, a tree $T_U \subseteq G$ containing all terminals of U, such that the trees $\{T_U\}_{U \in \mathcal{G}}$ are edge-disjoint.

Consider some group $U \in \mathcal{G}$ and its corresponding tree T_U. We construct a new tree T'_U, which is a minor of T_U, as follows. Root T_U at some arbitrary vertex r_U. While T_U contains a leaf v that does not belong to U, delete v from T_U. Assume now that every leaf of T_U belongs to U. While T_U contains any degree-2 vertex $v' \neq r_U$, we replace the two edges incident on v' with a single edge. Let T'_U be the resulting tree. It is easy to see that T'_U is a minor of T_U, and it contains at most $2|U|$ vertices, since its leaves belong to U. In order to construct the sparsifier H, we start with disjoint copies of trees T'_U (so if any vertex is contained in several such trees, we use several copies of this vertex). Finally, we add a new vertex r, and an edge (r, r_U) for every $U \in \mathcal{G}$, connecting r to the root of the tree T'_U. We claim that graph H is a quality-$(z, 31)$ integral flow sparsifier for G.

Indeed, let D be any set of demands over $\mathcal{T}$. By scaling D appropriately, we can assume without loss of generality that $\eta(G, D) = 1$. Let D' be the demand set obtained from D by scaling all demands down by factor z. We show that $\eta(H, D') \leq 1$. For each group $U \in \mathcal{G}$, the total demand originating from the terminals of U is at most 1. For each pair $t, t' \in U$, we route the demand $D'(t, t')$ along the tree T'_U. For each pair (t, t') with $t \in U$, $t' \in U'$, where $U \neq U'$, we route $D'(t, t')$ flow units from t to r_U along the tree T'_U, then use the edges (r_U, r) and $(r, r_{U'})$, and finally we route $D'(t, t')$ flow units from $r_{U'}$ to t' along the tree $T'_{U'}$. This gives a routing of D' with congestion at most 1. Therefore, $\eta(H, D) \leq z$.

Assume now that we are given some collection $\mathcal{M}$ of pairs of terminals and a set $\mathcal{P}$ of paths that connects the pairs of terminals in $\mathcal{M}$ with congestion at most η in graph H. We show a collection $\mathcal{P}'$ of paths connecting the same pairs of terminals with congestion at most 31η in graph G.

We decompose $\mathcal{M}$ into two subsets: $\mathcal{M}_1 \subseteq \mathcal{M}$ containing pairs (s, t) where both s and t belong to the same group U, and $\mathcal{M}_2 = \mathcal{M} \setminus \mathcal{M}_1$. For each group $U \in \mathcal{G}$, let $\mathcal{M}_1(U) \subseteq \mathcal{M}_1$ be the set of pairs that belong to group U. Then we can assume without loss of generality that all pairs in $\mathcal{M}_1(U)$ are routed along the tree T'_U in H, and the total congestion of this routing is at most η. We can then route these pairs along the tree T_U in graph G. We therefore obtain an integral routing of all pairs in $\mathcal{M}_1$ with congestion at most η in graph G.

We now turn to pairs in $\mathcal{M}_2$. Since the pairs in $\mathcal{M}_2$ can be routed in graph H with congestion at most η, for each $U \in \mathcal{G}$, the terminals of U participate in at most η pairs in $\mathcal{M}_2$. We decompose $\mathcal{M}_2$ into 2η subsets $\mathcal{M}^1, \ldots, \mathcal{M}^{2\eta}$, such that for each $1 \leq j \leq 2\eta$, for each $U \in \mathcal{G}$, at most one terminal of U participates in pairs in $\mathcal{M}^j$. Such a decomposition can be found greedily. We consider each pair $(s,t) \in \mathcal{M}_2$ in turn. Assume that $s \in U$, $t \in U'$. We select any index j, such that no terminal of $U \cup U'$ participates in any pair of $\mathcal{M}^j$, and add (s,t) to $\mathcal{M}^j$. Since for each $U \in \mathcal{G}$, the terminals of U participate in at most η pairs in $\mathcal{M}_2$, it is easy to see that this greedy process will give the desired decomposition. For each $1 \leq j \leq 2\eta$, the pairs in $\mathcal{M}^j$ now define a set D_j of $(1,\mathcal{G})$-restricted demands. Using Theorem 1.2, there is an efficient algorithm that with high probability finds a routing of D_j in G with congestion at most 15. Therefore, we obtain a routing of all pairs in $\mathcal{M}_2$ with congestion at most 30η. Overall, we route all pairs in $\mathcal{M}$ with congestion at most 31η. This concludes the proof that H is a quality-$(z,31)$ integral flow sparsifier for G. Notice that $|V(H)| \leq 2k$.

We now consider a general case where we are given a graph $G = (V,E)$, and set $\mathcal{T}$ of k terminals, such that the degree of every terminal is 1 in G, but $\mathcal{T}$ is not necessarily $\alpha_{\text{BW}}(k)$-well-linked in G. We compute a bandwidth-decomposition $\mathcal{W}$ of $V(G) \setminus \mathcal{T}$ using Corollary 2.9. Let G' be the graph obtained from G by subdividing every edge $e \in \bigcup_{W \in \mathcal{W}} \text{out}(W)$ by a vertex v_e. For each cluster $W \in \mathcal{W}$, let $\mathcal{T}_W = \{v_e \mid e \in \text{out}_G(W)\}$, and let $G_W = G'[W \cup \mathcal{T}_W]$. Notice that since W has the $\alpha_{\text{BW}}(k)$-bandwidth property, we are guaranteed that the set $\mathcal{T}_W$ of terminals is $\alpha_{\text{BW}}(k)$-well-linked in graph G_W. We can then compute a sparsifier H_W for G_W as before.

In order to obtain the final sparsifier H, we replace, for each $W \in \mathcal{W}$, graph G_W with graph H_W in G'. In order to do so, we delete all vertices of W from G', and add the vertices and the edges of H_W to it. Finally, for each $t \in \mathcal{T}_W$, we identify the two copies of t in the resulting graph. It is immediate to verify that the resulting graph H is a quality-$(z,31)$ integral flow sparsifier for $(G,\mathcal{T})$, using the fact that for each $W \in \mathcal{W}$, graph H_W is a quality-$(z,31)$ integral flow sparsifier for $(G_W, \mathcal{T}_W)$.

Appendix A. Proofs omitted from section 2.

A.1. Proof of Theorem 2.1. Let $\ell = 4d\beta(n)/\alpha$, where $\beta(n) = O(\log n)$ is the flow-cut gap for undirected graphs. The algorithm greedily selects a source-sink pair (s_i, t_i) that has a path P of length at most ℓ connecting s_i to t_i in the current graph G. We then remove all vertices of P from the graph G and continue. The algorithm terminates when for each remaining source-sink pair (s_i, t_i), every path connecting s_i to t_i has length at least ℓ.

Note that in each iteration of the algorithm, we route one demand pair and remove at most $(\ell+1)d$ edges from the graph. The key to the algorithm analysis is to show that when the algorithm terminates, we have removed many edges from the graph, and therefore we have routed many of the demand pairs.

Let E' be the subset of edges removed from the graph by the algorithm, and let E'' be the subset of remaining edges. We first claim that there is a multicut in graph G whose value is at most $|E'| + |E''| \cdot \beta(n)/\ell$. Indeed, let $G' = G[E'']$ be the graph obtained when the algorithm terminates, and let $\mathcal{M}'$ be the set of the surviving source-sink pairs. Consider the instance of the multicut problem on graph G' with the set $\mathcal{M}'$ of demand pairs. Setting the weight of each edge in E'' to $1/\ell$, we obtain a feasible fractional solution to this multicut instance, since the length of every path connecting every pair of terminals is at least ℓ. Therefore, there is an integral solution to this multicut instance of value $|E''| \cdot \beta(n)/\ell$. Adding the subset E' of edges, we

obtain a feasible solution to the multicut problem on the original graph G of value $|E'| + |E''| \cdot \beta(n)/\ell$. (Notice that all edges incident on the terminals that participate in pairs in $\mathcal{M} \setminus \mathcal{M}'$ belong to E'.)

On the other hand, the value of any multicut in graph G is at least $\alpha|V|/2$. Indeed, if E^* is any feasible solution to the multicut problem, then each connected component C of $G \setminus E^*$ contains at most $|V|/2$ vertices and therefore has at least $\alpha|V(C)|$ outgoing edges. Since each edge is counted at most twice, we get that $|E^*| \geq \alpha|V|/2$.

We conclude that $|E'| + |E''| \cdot \beta(n)/\ell \geq \alpha|V|/2$, and so

$$|E'| \geq \frac{\alpha|V|}{2} - \frac{|E''| \cdot \beta(n)}{\ell} \geq \frac{\alpha|V|}{2} - \frac{|E| \cdot \beta(n)}{\ell} \geq \frac{\alpha|V|}{2} - \frac{d|V|\beta(n)}{2\ell} \geq \frac{\alpha|V|}{4}$$

since $\ell = 4d\beta(n)/\alpha$. Therefore, at least $\alpha|V|/4$ edges have been deleted from the graph. Since in each iteration we only delete at most $d(\ell+1)$ edges, overall the number of pairs routed is at least $\frac{\alpha|V|}{4d(\ell+1)} = \Omega(\frac{\alpha^2|V|}{d^2 \log n})$.

A.2. Proof of Theorem 2.10. Let T be any spanning tree of the graph G, and assume that it is rooted at some vertex r. We perform a number of iterations, where in each iteration we delete some edges and vertices from T. For each vertex v of the tree T, let T_v denote the subtree rooted at v, and let $w(T_v)$ denote the total weight of all vertices in T_v. We build the partition $\mathcal{G}$ of V gradually. At the beginning, $\mathcal{G} = \emptyset$. While $w(T) > 3p$, we perform the following iteration:

- Let v be the lowest vertex in the tree T, such that $w(T_v) > p$.
- If $w(T_v) \leq 2p$, then we add a new group U to $\mathcal{G}$, containing all vertices of T_v, and we delete T_v from the tree T, setting $T_U = T_v$.
- Otherwise, let $u_1, \ldots, u_k$ be the children of v, and let j be the smallest index, such that $\sum_{i=1}^{j} w(T_{u_i}) \geq p$. We add a new group U to $\mathcal{G}$, consisting of all vertices in trees $T_{u_1}, \ldots, T_{u_j}$. Notice that $w(U) \leq 2p$ must hold. We let T_U be the subtree of T consisting of v and the trees $T_{u_1}, \ldots, T_{u_j}$. We delete the trees $T_{u_1}, \ldots, T_{u_j}$ from the tree T.

Notice that if, at the beginning of the current iteration, $w(T) > 3p$, then at the end of the current iteration, $w(T) > p$ must hold. In the last iteration, when $w(T) \leq 3p$, we add a final group U to $\mathcal{G}$, containing all vertices currently in the tree T, and we let T_U be the current tree T. It is easy to verify that all conditions of the theorem hold for the final partition $\mathcal{G}$ of V.

Appendix B. Proof of Theorem 3.1.

For the proof of this theorem, we need a more general definition of well-linkedness that was used in [17]. Suppose we are given a graph $G = (V, E)$, and for each vertex $v \in V$, we are given a weight $\pi(v)$. For a subset $S \subseteq V$ of vertices, let $\pi(S) = \sum_{v \in S} \pi(v)$. We say that G is π-flow well-linked iff each pair (u, v) of vertices can simultaneously send $\frac{\pi(u) \cdot \pi(v)}{\pi(V)}$ flow units to each other with no congestion. We start with the following theorem, that was proved in [17], using a flow-well-linked graph decomposition.

THEOREM B.1 (Theorem 2.1 in [17]). *Let $G = (V, E)$ be any graph, and let $\mathcal{M}$ be a set of k source-sink pairs in G. We can efficiently find a partition $G_1, \ldots, G_\ell$ of G into vertex-disjoint induced subgraphs, and for each $1 \leq i \leq \ell$, find a weight function $\pi_i : V(G_i) \to \mathbb{R}^+$ with the following properties. Let $\mathcal{M}'_i \subseteq \mathcal{M}$ be the set of source-sink pairs contained in G_i, and let $\mathcal{T}'_i$ be the set of all terminals participating in $\mathcal{M}'_i$. Then the following hold.*

- *For all $1 \leq i \leq \ell$:*

  - *for all* $u \in \mathcal{T}'_i$, $\pi_i(u) \leq 1$.
  - *for all* $(u,v) \in \mathcal{M}'_i$, $\pi_i(u) = \pi_i(v)$.
  - *Graph* G_i *is* π_i*-flow well-linked.*
- $\sum_{i=1}^{\ell} \pi_i(\mathcal{T}'_i) = \Omega(\mathsf{OPT}/(\beta(k) \cdot \log \mathsf{OPT})) = \Omega(\mathsf{OPT}/\log^2 k)$.

In order to complete the proof of the theorem, it is enough to show that we can find, for each $1 \leq i \leq \ell$, a subset $\mathcal{M}_i \subseteq \mathcal{M}'_i$ of source-sink pairs, with $|\mathcal{M}_i| = \Omega(\pi_i(\mathcal{T}'_i))$, such that the set $\mathcal{T}_i$ of all terminals participating in pairs in $\mathcal{M}_i$ is $\frac{1}{2}$-well-linked in G_i.

Fix some $1 \leq i \leq \ell$. We find a grouping $\mathcal{G}_i$ of the terminals in set $\mathcal{T}'_i$, using the weights π_i and the grouping parameter $p = 2$, as in Theorem 2.10, so for each group $U \in \mathcal{G}_i$, $2 \leq \pi_i(U) \leq 6$. Next, we will gradually construct the set $\mathcal{M}_i$ of source-sink pairs, starting from $\mathcal{M}_i = \emptyset$. In each iteration, we will add one source-sink pair to $\mathcal{M}_i$ and remove some source-sink pairs from $\mathcal{M}'_i$, charging their weights to the pair that was added to $\mathcal{M}_i$. While $\mathcal{M}'_i$ is nonempty, we perform the following procedure:

- Let $(s,t) \in \mathcal{M}'_i$ be any source-sink pair. Add (s,t) to $\mathcal{M}_i$.
- If both s and t belong to the same group $U \in \mathcal{G}$, then for each pair $(u,v) \in \mathcal{M}'_i$, where $u \in U$ or $v \in U$, remove (u,v) from $\mathcal{M}'_i$ and charge the weight $\pi_i(v)$ and $\pi_i(v)$ to (s,t). Notice that the total weight charged to (s,t) is at most 12.
- Otherwise, let U_1 be the group to which s belongs, and let U_2 be the group to which t belongs. For each pair $(u,v) \in \mathcal{M}'_i$, such that either $u \in U_1 \cup U_2$ or $v \in U_1 \cup U_2$, remove (u,v) from $\mathcal{M}'_i$ and charge the weights $\pi_i(u)$ and $\pi_i(v)$ to (s,t). Notice that the total weight charged to (s,t) in this step is at most 24.

The procedure stops when $\mathcal{M}'_i = \emptyset$. Let $\mathcal{M}_i$ be the resulting set of source-sink pairs, and let $\mathcal{T}_i$ be the set of terminals participating in them. From the above charging scheme, it is clear that $|\mathcal{M}_i| = \Omega(\pi_i(\mathcal{T}'_i))$, as required. Observe also that for each group $U \in \mathcal{G}_i$, at most one terminal $v \in U$ belongs to $\mathcal{T}_i$. Finally, we need to show that $\mathcal{T}_i$ is $\frac{1}{2}$-well-linked in G_i. For each vertex $v \in \mathcal{T}_i$, let $U_v \in \mathcal{G}_i$ be the group to which v belongs.

Let (A,B) be any partition of $V(G_i)$, and assume without loss of generality that $|A \cap \mathcal{T}_i| \geq |B \cap \mathcal{T}_i| = k'$. Let $\mathcal{T}_A$ be any subset of k' vertices of $A \cap \mathcal{T}_i$, and let $\mathcal{T}_B = B \cap \mathcal{T}_i$. Let $\mathcal{M}^*$ be any matching between the vertices of $\mathcal{T}_A$ and the vertices of $\mathcal{T}_B$. We show how to route this matching with congestion at most 2 in G_i, proving that $|E_{G_i}(A,B)| \geq k'/2$, as required.

We find the routing in two steps. In the first step, we construct a flow F_1, where for each pair $(v,v') \in \mathcal{M}^*$, the vertices in U_v send 1 flow unit in total to the vertices in $U_{v'}$, each vertex $x \in U_v$ sends at most $\pi_i(x)$ flow units, and each vertex $y \in U_{v'}$ receives at most $\pi_i(y)$ flow units, with total congestion at most 1. This flow is defined as follows. Recall that graph G_i is π_i-well-linked. Therefore, every pair (x,y) of vertices can send $\frac{\pi_i(x) \cdot \pi_i(y)}{\pi_i(V(G_i))}$ flow units to each other with no congestion. Let F denote this flow. Fix some pair $(v,v') \in \mathcal{M}^*$. In flow F, there are $\pi_i(U_v)$ flow units originating from the vertices in U_v that are then distributed among the vertices of G, and the amount of flow each vertex z of G receives is $\pi_i(z) \cdot \pi_i(U_v)/\pi_i(V(G_i))$. We scale the flow originating from the vertices in U_v down by factor $\pi_i(U_v)/2$, so that every vertex z of G now receives $2\pi_i(z)/\pi_i(V(G_i))$ flow units from U_v. We

perform a similar transformation for the flow originating at the vertices of $U_{v'}$, and we concatenate both flows. As a result, we obtain a flow where the vertices in U_v send two flow units in total to the vertices in $U_{v'}$. Taking the union of these flows over all $(v, v') \in \mathcal{M}^*$, and scaling them down by factor 2, gives us the flow F_1. It is easy to see that the total congestion caused by F_1 is at most 1. This is since each flow-path in F is used at most twice: once for each of its endpoints. Finally, in order to route the matching $\mathcal{M}^*$, consider any pair $(v, v') \in \mathcal{M}^*$. Vertex v will distribute one flow unit to the vertices in U_v, along the tree T_{U_v}, where the amount of flow each vertex $x \in T_{U_v}$ receives equals the amount of flow it sends out in F_1. We then use the flow F_1 to route this one flow unit to the vertices of $U_{v'}$. Finally, vertex v' collects one flow unit from the vertices of $U_{v'}$ along the tree $T_{U_{v'}}$. It is easy to see that the total congestion caused by this flow is at most 2, since all trees $\{T_U\}_{U \in \mathcal{G}_i}$ are edge-disjoint. We conclude that $|E_{G_i}(A, B)| \geq k'/2$, and $\mathcal{T}_i$ is $\frac{1}{2}$-well-linked in G_i.

Appendix C. Proof of Theorem 4.1.

We use the result of Leighton and Rao [46], who have shown that any demand that is routable on an expander graph with no congestion can also be routed on relatively short paths with small congestion. Specifically, the following is a slightly rephrased statement of Theorem 18 from [46] and its immediate corollary.

THEOREM C.1 (Theorem 18 from [46]). *Let G be any n-vertex α-expander with maximum vertex degree $d_{\max}$. Then every pair of vertices in G can send $\Omega(\alpha/(n \log n))$ flow units to each other with no congestion, on flow-paths of length $O(d_{\max} \log n/\alpha)$. Moreover, such flow can be found efficiently.*

COROLLARY C.2. *Let G be any n-vertex α-expander with maximum vertex degree $d_{\max}$, and let M be any partial matching over the vertices of G. Then there is an efficient randomized algorithm that finds, for every pair $(u, v) \in M$, a set $\mathcal{P}_{u,v}$ of $h = \lceil \log n \rceil$ paths of length $O(d_{\max} \log n/\alpha)$ each, such that the set $\mathcal{P} = \bigcup_{(u,v) \in M} \mathcal{P}_{u,v}$ of paths causes congestion $O(\log^2 n/\alpha)$ in G. The algorithm succeeds with high probability.*

Proof. We start by showing that there is a multicommodity flow f, where every pair $(u, v) \in M$ of vertices sends one flow unit to each other simultaneously, on flow-paths of length $O(d_{\max} \log n/\alpha)$, with total congestion $O(\log n/\alpha)$. Let f' be the flow guaranteed by Theorem C.1, scaled up by factor $O(\log n/\alpha)$, so that every pair of vertices now sends $1/n$ flow units to each other, with total congestion $O(\log n/\alpha)$. Let $(u, v) \in M$ be any pair of vertices. The flow between u and v is defined as follows: u will send $1/n$ flow units to each vertex of G, using the flow f', and v will collect $1/n$ flow units from each vertex in G, using the flow f'. In other words, the flow f between u and v is obtained by concatenating all flow-paths in f' originating at u with all flow-paths in f' terminating at v. It is easy to see then that every flow-path in f' is used at most twice: once by each of its endpoints; all flow-paths in f have length $O(d_{\max} \log n/\alpha)$, and the total congestion of flow f is $O(\log n/\alpha)$.

In order to find the sets $\mathcal{P}_{u,v}$ of paths for each pair $(u, v) \in M$, we perform the standard randomized rounding: for each pair $(u, v) \in M$, we perform h independent random trials. In each trial, we randomly choose one of the flow-paths connecting u to v, with probability equal to the amount of flow sent via this path in f, and add this path to $\mathcal{P}_{u,v}$. This ensures that all paths in set $\mathcal{P}_{u,v}$ have length $O(d_{\max} \log n/\alpha)$. The expected congestion on any edge is $O(h \log n/\alpha)$, and using the standard Chernoff

bounds, it is easy to see that with high probability, the congestion on every edge is $O(\log^2 n/\alpha)$. □

We are now ready to prove Theorem 4.1.

Proof of Theorem 4.1. Let $L = O(d_{\max} \log n/\alpha)$ be the bound on the path length, and let $\eta = O(\log^2 n/\alpha)$ be the bound on the congestion guaranteed by Corollary C.2. We set $m = \frac{(4e\cdot\eta^{c+2}\cdot d_{\max}^{c+2}\cdot L)^{1/c}}{h} = O(d_{\max}^{1+3/c}(\log n)^{1+5/c}/\alpha^{1+3/c})$.

Given the matching $M \subseteq ([r] \times [r])$, we assume without loss of generality that $M = \{(2i-1, 2i)\}_{i=1}^{\lfloor r/2 \rfloor}$. We extend the matching M to the vertices of G, by defining a matching M' as follows. For each $1 \leq i \leq \lfloor r/2 \rfloor$, we select any maximal matching M_i between the vertices of V_{2i-1} and the vertices of V_{2i} and add it to M'. Notice that M_i must contain at least m pairs of vertices. Next, we use Corollary C.2 to find a collection $\mathcal{P}$ of paths of length at most L each that cause a total congestion of at most η in G, such that for each $(u, v) \in M'$, there is a set $\mathcal{P}_{u,v}$ of h paths connecting u to v in $\mathcal{P}$. For each $i : 1 \leq i \leq \lfloor r/2 \rfloor$, we let B_i denote the union of the sets $\mathcal{P}_{u,v}$ of paths, for $u \in V_{2i-1}$, $v \in V_{2i}$, and we call B_i the ith bundle. Notice that $|B_i| \geq mh$. If B_i contains more than mh paths, we discard paths from B_i until $|B_i| = mh$ holds for all i.

Finally, we will select one path from each bundle, such that the resulting set of paths causes vertex congestion at most c. We do so using the constructive version of the Lovasz local lemma by Moser and Tardos [50]. The next theorem summarizes the symmetric version of the result of [50].

THEOREM C.3 (see [50]). *Let X be a finite set of mutually independent random variables in some probability space. Let $\mathcal{A}$ be a finite set of bad events determined by these variables. For each event $A \in \mathcal{A}$, let $vbl(A) \subseteq X$ be the unique minimal subset of variables determining A, and let $\Gamma(A) \subseteq \mathcal{A}$ be a subset of bad events B, such that $A \neq B$, but $vbl(A) \cap vbl(B) \neq \emptyset$. Assume further that for each $A \in \mathcal{A}$, $|\Gamma(A)| \leq D$, $\mathbf{Pr}[A] \leq p$, and $ep(D+1) \leq 1$. Then there is an efficient randomized algorithm that with high probability finds an assignment to the variables of X, such that none of the events in $\mathcal{A}$ holds.*

For each bundle B_i we choose one of its paths P_i independently at random. We let x_i be the variable indicating which path has been chosen. For each vertex $v \in V$, we let β_v be the bad event that v belongs to more than c of the chosen paths. Since the congestion on every edge due to paths in $\mathcal{P}$ is at most η, and the maximum vertex degree is $d_{\max}$, we get that there are at most $(\eta d_{\max})^{c+1}$ potential $(c+1)$-tuples of paths containing v (where we only consider $(c+1)$-tuples containing at most one path from each bundle), and each tuple is chosen with probability $1/(hm)^{c+1}$. Therefore, $\mathbf{Pr}[\beta_v] \leq \frac{(\eta d_{\max})^{c+1}}{(hm)^{c+1}}$. We denote $p = (\frac{\eta d_{\max}}{hm})^{c+1}$.

The set $vbl(\beta_v)$ of variables contains all variables x_i, where the bundle B_i contains a path $P \in \mathcal{P}$, such that $v \in P$. Therefore, $|vbl(\beta_v)| \leq \eta d_{\max}$. For each such variable x_i, there are mh paths participating in the bundle B_i, each of which contains at most $(L+1)$ vertices. Therefore, $|\Gamma(\beta_v)| \leq mh(L+1)\eta d_{\max}$. We denote this value by D.

It now only remains to show that $(D+1)ep \leq 1$, which follows from the choice of $m = \frac{(4e\cdot\eta^{c+2}\cdot d_{\max}^{c+2}\cdot L)^{1/c}}{h}$. □

Appendix D. Table of parameters.

$\gamma_{\text{CMG}}(k)$	$\Theta(\log^2 k)$	Parameter from the cut-matching game of [39], from Theorem 2.2. Is also denoted by γ.
$\alpha_{\text{ARV}}(k)$	$O(\sqrt{\log k})$	Approximation factor of the algorithm of [6] for Sparsest Cut.
$\alpha(k)$	$\frac{1}{2^{11}\cdot\gamma_{\text{CMG}}(k)\cdot\log k} = \Omega\left(\frac{1}{\log^3 k}\right)$	Well-linkedness parameter from Theorem 2.8.
$\alpha_{\text{BW}}(k)$	$\alpha(k)/\alpha_{\text{ARV}}(k) = \Omega\left(\frac{1}{\log^{3.5} k}\right)$	Well-linkedness parameter from Corollary 2.9.
$\beta(k)$	$\Theta(\log k)$	Flow-cut gap for concurrent flow on k terminals.
k_1	$\frac{k}{192\gamma^3\log\gamma} = \Omega\left(\frac{k}{\log^6 k\log\log k}\right)$	Parameter from the definition of legal contracted graphs.
p	$\frac{8\beta(k)}{\alpha_{\text{BW}}(k)} = O(\log^{4.5} k)$	Grouping parameter for the sets Γ_j.
k'	$\lfloor\frac{1}{2\gamma^3}\cdot\lfloor\frac{k_1}{6p}\rfloor\rfloor = \Omega\left(\frac{k}{\log^{16.5} k\log\log k}\right)$	Number of vertices in the expander X.
k^*	$\lfloor\frac{k_1}{6p}\rfloor$	Size of sets Γ'_j (that contain at most one edge from each group of $\mathcal{G}_j$).

Acknowledgments. The author thanks Matthew Andrews and Sanjeev Khanna for many inspiring discussions about routing problems. She also thanks the anonymous reviewers for many helpful comments.

REFERENCES

[1] M. Andrews, *Approximation algorithms for the edge-disjoint paths problem via Raecke decompositions*, in Proceedings of the 51st IEEE Annual Symposium on Foundations of Computer Science, FOCS '10, IEEE Computer Society, Washington, DC, 2010, pp. 277–286.

[2] 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, 30 (2010), pp. 485–520.

[3] M. Andrews and L. Zhang, *Hardness of the undirected edge-disjoint paths problem*, in Proceedings of STOC, Harold N. Gabow and Ronald Fagin, eds., ACM, New York, 2005, pp. 276–283.

[4] M. Andrews and L. Zhang, *Hardness of the undirected congestion minimization problem*, SIAM J. Comput., 37 (2007), pp. 112–131.

[5] M. Andrews and L. Zhang, *Almost-tight hardness of directed congestion minimization*, J. ACM, 55 (2008), 27.

[6] S. Arora, S. Rao, and U. V. Vazirani, *Expander flows, geometric embeddings and graph partitioning*, J. ACM, 56 (2009), 5.

[7] Y. Azar and O. Regev, *Combinatorial algorithms for the unsplittable flow problem*, Algorithmica, 44 (2006), pp. 49–66.

[8] A. Baveja and A. Srinivasan, *Approximation algorithms for disjoint paths and related routing and packing problems*, Math. Oper. Res., 25 (2000), pp. 255–280.

[9] A. Z. Broder, A. M. Frieze, S. Suen, and E. Upfal, *Optimal construction of edge-disjoint paths in random graphs*, in Proceedings of the 5th ACM-SIAM SODA, ACM, New York, SIAM, Philadelphia, 1994, pp. 603–612.

[10] A. Z. Broder, A. M. Frieze, and E. Upfal, *Existence and construction of edge-disjoint paths on expander graphs*, SIAM J. Comput., 23 (1994), pp. 976–989.

[11] M. Charikar, T. Leighton, S. Li, and A. Moitra, *Vertex sparsifiers and abstract rounding algorithms*, in Proceedings of the 51st IEEE Annual Symposium on Foundations of Computer Science, FOCS '10, IEEE Computer Society, Washington, DC, 2010, pp. 265–274.

[12] C. Chekuri and J. Chuzhoy, *Large-treewidth graph decompositions and applications*, in Proceedings of ACM STOC, ACM, New York, 2013, pp. 291–300.

[13] C. Chekuri and J. Chuzhoy, *Polynomial bounds for the grid-minor theorem*, in Proceedings of the 46th ACM STOC, ACM, New York, 2014, pp. 60–69.

[14] C. Chekuri and J. Chuzhoy, *Degree-3 treewidth sparsifiers*, in Proceedings of the 26th ACM-SIAM SODA, ACM, New York, SIAM, Philadelphia, 2015, pp. 242–255.

[15] C. Chekuri and A. Ene, *Poly-logarithmic approximation for maximum node disjoint paths with constant congestion*, in Proceedings of the 24th ACM-SIAM SODA, ACM, New York, SIAM, Philadelphia, 2013, pp. 326–341.
[16] C. Chekuri and S. Khanna, *Edge-disjoint paths revisited*, ACM Trans. Algorithms, 3 (2007), 46.
[17] C. Chekuri, S. Khanna, and F. B. Shepherd, *Multicommodity flow, well-linked terminals, and routing problems*, in Proceedings of the 37th ACM STOC, ACM, New York, 2005, pp. 183–192.
[18] C. Chekuri, S. Khanna, and F. B. Shepherd, *An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow*, Theory Comput., 2 (2006), pp. 137–146.
[19] C. Chekuri, S. Khanna, and F. B. Shepherd, *Edge-disjoint paths in planar graphs with constant congestion*, SIAM J. Comput., 39 (2009), pp. 281–301.
[20] C. Chekuri, S. Khanna, and F. B. Shepherd, *The all-or-nothing multicommodity flow problem*, SIAM J. Comput., 42 (2013), pp. 1467–1493.
[21] C. Chekuri, M. Mydlarz, and F. B. Shepherd, *Multicommodity demand flow in a tree and packing integer programs*, ACM Trans. Algorithms, 3 (2007), 27.
[22] J. Chuzhoy, *On vertex sparsifiers with Steiner nodes*, in Proceedings of the 44th ACM STOC, H. J. Karloff and T. Pitassi, eds., ACM, New York, 2012, pp. 673–688.
[23] J. Chuzhoy, V. Guruswami, S. Khanna, and K. Talwar, *Hardness of routing with congestion in directed graphs*, in Proceedings of ACM STOC, ACM, New York, 2007, pp. 165–178.
[24] J. Chuzhoy and S. Li, *A polylogarithmic approximation algorithm for edge-disjoint paths with congestion* 2, in Proceedings of IEEE FOCS, IEEE Computer Society, Washington, DC, 2012, pp. 233–242.
[25] D. Dubhashi and A. Panconesi, *Concentration of Measure for the Analysis of Randomized Algorithms*, Cambridge University Press, New York, 2009.
[26] M. Englert, A. Gupta, R. Krauthgamer, H. Räcke, I. Talgam-Cohen, and K. Talwar, *Vertex sparsifiers: New results from old techniques*, SIAM J. Comput., 43 (2014), pp. 1239–1262.
[27] S. Even, A. Itai, and A. Shamir, *On the complexity of timetable and multicommodity flow problems*, SIAM J. Comput., 5 (1976), pp. 691–703.
[28] A. Frank, *Edge-disjoint paths in planar graphs*, J. Combin. Theory, 39 (1985), pp. 164–178.
[29] A. Frank, *On connectivity properties of Eulerian digraphs*, in Graph Theory in Memory of G. A. Dirac (Sandbjerg, 1985), Ann. Discrete Math. 41, North–Holland, Amsterdam, 1989, pp. 179–194.
[30] A. M. Frieze, *Edge-disjoint paths in expander graphs*, SIAM J. Comput., 30 (2001), pp. 1790–1801.
[31] N. Garg, V. V. Vazirani, and M. Yannakakis, *Primal-dual approximation algorithms for integral flow and multicut in trees*, Algorithmica, 18 (1997), pp. 3–20.
[32] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis, *Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems*, J. Comput. System Sci., 67 (2003), pp. 473–496.
[33] A. Hajnal and E. Szemerédi, *Proof of a conjecture of P. Erdös*, in Proceedings of a Colloquium on Combinatorial Theory and Its Applications II, North–Holland, Amsterdam, 1970, pp. 601–623.
[34] B. Jackson, *Some remarks on arc-connectivity, vertex splitting, and orientation in graphs and digraphs*, J. Graph Theory, 12 (1998), pp. 429–436.
[35] D. R. Karger, *Random sampling in cut, flow, and network design problems*, Math. Oper. Res., 24 (1999), pp. 383–413.
[36] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. Miller and J. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
[37] K. Kawarabayashi and Y. Kobayashi, *Breaking $O(n^{1/2})$-approximation algorithms for the edge-disjoint paths problem with congestion two*, in Proceedings of ACM STOC, L. Fortnow and S. P. Vadhan, eds., ACM, New York, 2011, pp. 81–88.
[38] K. Kawarabayashi and Y. Kobayashi, *An $O(\log n)$-approximation algorithm for the edge-disjoint paths problem in Eulerian planar graphs*, ACM Trans. Algorithms, 9 (2013), 16.
[39] R. Khandekar, S. Rao, and U. V. Vazirani, *Graph partitioning using single commodity flows*, J. ACM, 56 (2009), 19.
[40] J. Kleinberg, *Approximation Algorithms for Disjoint Paths Problems*, Ph.D. thesis, MIT, Cambridge, MA, 1996.
[41] J. Kleinberg and R. Rubinfeld, *Short paths in expander graphs*, in Proceedings of the 37th IEEE FOCS, IEEE, Washington, DC, 1996, pp. 86–95.

[42] J. M. Kleinberg, *An approximation algorithm for the disjoint paths problem in even-degree planar graphs.*, in Proceedings of IEEE FOCS, IEEE, Washington, DC, 2005, pp. 627–636.
[43] J. M. Kleinberg and É. Tardos, *Approximations for the disjoint paths problem in high-diameter planar networks*, J. Comput. System Sci., 57 (1998), pp. 61–73.
[44] S. G. Kolliopoulos and C. Stein, *Approximating disjoint-path problems using packing integer programs*, Math. Program., 99 (2004), pp. 63–87.
[45] F. T. Leighton and A. Moitra, *Extensions and limits to vertex sparsification*, in Proceedings of the 42nd ACM STOC, ACM, New York, 2010, pp. 47–56.
[46] F. T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.
[47] W. Mader, *A reduction method for edge connectivity in graphs*, Ann. Discrete Math., 3 (1978), pp. 145–164.
[48] K. Makarychev and Y. Makarychev, *Metric extension operators, vertex sparsifiers and Lipschitz extendability*, in Proceedings of IEEE FOCS, IEEE Computer Society, Washington, DC, 2010, pp. 255–264.
[49] A. Moitra, *Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size*, in Proceedings of IEEE FOCS, IEEE Computer Society, Washington, DC, 2009, pp. 3–12.
[50] R. A. Moser and G. Tardos, *A constructive proof of the general Lovász local lemma*, J. ACM, 57 (2010), 11.
[51] L. Orecchia, L. J. Schulman, U. V. Vazirani, and N. K. Vishnoi, *On partitioning graphs via single commodity flows*, in Proceedings of the 40th ACM STOC, ACM, New York, 2008, pp. 461–470.
[52] H. Räcke, *Minimizing congestion in general networks*, in Proceedings of the 43rd IEEE FOCS, IEEE, Washington, DC, 2002, pp. 43–52.
[53] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.
[54] S. Rao and S. Zhou, *Edge disjoint paths in moderately connected graphs*, SIAM J. Comput., 39 (2010), pp. 1856–1887.
[55] N. Robertson and P. D. Seymour, *An outline of a disjoint paths algorithm*, in Paths, Flows, and VLSI-Layout, Springer-Verlag, Berlin, 1990, pp. 267–292.
[56] N. Robertson and P. D. Seymour, *Graph minors.* XIII. *The disjoint paths problem*, J. Combin. Theory Ser. B, 63 (1995), pp. 65–110.
[57] L. Seguin-Charbonneau and F. B. Shepherd, *Maximum edge-disjoint paths in planar graphs with congestion* 2, in Proceedings of IEEE FOCS, R. Ostrovsky, ed., IEEE, Washington, DC, 2011, pp. 200–209.
[58] M. Singh and L. C. Lau, *Approximating minimum bounded degree spanning trees to within one of optimal*, J. ACM, 62 (2015), 1.
[59] A. Srinivasan, *Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems*, in Proceedings of IEEE FOCS, IEEE, Washington, DC, 1997, pp. 416–425.
[60] K. R. Varadarajan and G. Venkataraman, *Graph decomposition and a greedy algorithm for edge-disjoint paths*, in Proceedings of ACM-SIAM SODA, ACM, New York, SIAM, Philadelphia, 2004, pp. 379–380.