A Distanced Matching Game, Decremental APSP in Expanders, and Faster Deterministic Algorithms for Graph Cut Problems*

Julia Chuzhoy†

November 6, 2022

Abstract

Expander graphs play a central role in graph theory and algorithms. With a number of powerful algorithmic tools developed around them, such as the Cut-Matching game, expander pruning, expander decomposition, and algorithms for decremental All-Pairs Shortest Paths (APSP) in expanders, to name just a few, the use of expanders in the design of graph algorithms has become ubiquitous. Specific applications of interest to us are fast deterministic algorithms for cut problems in static graphs, and algorithms for dynamic distance-based graph problems, such as APSP.

Unfortunately, the use of expanders in these settings incurs a number of drawbacks. For example, the best currently known algorithm for decremental APSP in constant-degree expanders can only achieve a $(\log n)^{O(1/\epsilon^2)}$-approximation with $n^{1+O(\epsilon)}$ total update time for any ϵ. All currently known algorithms for the Cut Player in the Cut-Matching game are either randomized, or provide rather weak guarantees: expansion $1/(\log n)^{1/\epsilon}$ with running time $n^{1+O(\epsilon)}$. This, in turn, leads to somewhat weak algorithmic guarantees for several central cut problems: the best current almost linear time deterministic algorithms for Sparsest Cut, Lowest Conductance Cut, and Balanced Cut can only achieve approximation factor $(\log n)^{\omega(1)}$. Lastly, when relying on expanders in distance-based problems, such as dynamic APSP, via current methods, it seems inevitable that one has to settle for approximation factors that are at least $\Omega(\log n)$. In contrast, we do not have any negative results that rule out a factor-5 approximation with near-linear total update time.

In this paper we propose the use of well-connected graphs, and introduce a new algorithmic toolkit for such graphs that, in a sense, mirrors the above mentioned algorithmic tools for expanders. One of these new tools is the Distanced Matching game, an analogue of the Cut-Matching game for well-connected graphs. We demonstrate the power of these new tools by obtaining better results for several of the problems mentioned above. First, we design an algorithm for decremental APSP in expanders with significantly better guarantees: in a constant-degree expander, the algorithm achieves $(\log n)^{1+o(1)}$-approximation, with total update time $n^{1+o(1)}$. We also obtain a deterministic algorithm for the Cut Player in the Cut-Matching game that achieves expansion $\frac{1}{(\log n)^{5+o(1)}}$ in time $n^{1+o(1)}$, deterministic almost linear-time algorithms for Sparsest Cut, Lowest-Conductance Cut, and Minimum Balanced Cut with approximation factors $O(\text{poly}\log n)$, as well as improved deterministic algorithm for Expander Decomposition. We believe that the use of well-connected graphs instead of expanders in various dynamic distance-based problems (such as APSP in general graphs) has the potential of providing much stronger guarantees, since we are no longer necessarily restricted to superlogarithmic approximation factors.

*SODA 2023, to appear.

†Toyota Technological Institute at Chicago. Email: cjulia@ttic.edu. Supported in part by NSF grant CCF-2006464.

Contents

1 Introduction

Expander graphs are a central graph theoretic object that has been studied extensively, and they are frequently used in the design of graph algorithms. In recent years, a number of powerful algorithmic tools have been developed around expanders, including, for example, the Cut-Matching Game of [KRV09], expander pruning of [SW19], Expander Decomposition, and algorithms for decremental All-Pairs Shortest Paths (APSP) in expanders (see e.g. [CS21, Chu21]), to name just a few. This powerful algorithmic toolkit has led to many new algorithms for optimization problems in graphs. In this paper we are most interested in applications to fast deterministic algorithms for classical cut problems, such as Sparsest Cut, Lowest Conductance Cut, and Minimum Balanced Cut, and to dynamic algorithms, especially for distance-based graph problems, such as APSP.

Unfortunately, the use of expander graphs with the currently available expander-related tools incurs a number of drawbacks in these settings. While the results of [KRV09] provide a near-linear time randomized algorithm for the Cut Player in the Cut-Matching Game, which guarantees that the resulting graph is an $\Omega(1)$-expander, in the regime of deterministic algorithms, the best currently known results are significantly weaker: the algorithm of [CGL$^+$20] provides an implementation of the Cut Player in time $O\left(n^{1+O(\epsilon)}\right)$, but only guarantees expansion of $\frac{1}{(\log n)^{1/\epsilon}}$ for a sufficiently large ϵ. As a result, despite having randomized algorithms for Sparsest Cut, Lowest Conductance Cut, and Minimum Balanced Cut, that achieve $O(\log^2 n)$-approximation in $m^{1+o(1)}$ time, the best current deterministic algorithms for these problems with running time $m^{1+o(1)}$ can only achieve approximation factor $(\log n)^{\omega(1)}$. Similar issues, that we discuss in more detail below, arise in algorithms for Expander Decomposition. We note that all these cut problems are used extensively, including in settings (such as, e.g. dynamic algorithms), where deterministic algorithms are especially desirable. In recent years, the above mentioned expander-based tools have also found many applications in dynamic algorithms; for now we focus on decremental All-Pairs Shortest Paths (APSP). In this context, the use of expanders with the currently available algorithmic tools also has several drawbacks. First, in a typical use of expanders in dynamic APSP, one cannot obtain a better than $\Theta(\log n)$ approximation factor; we discuss this in more detail below. Second, the use of expanders usually involves either the Cut-Matching Game, or Expander Decomposition, which suffer from the drawbacks mentioned above. Lastly, in most uses of expander graphs for dynamic APSP, one needs to rely on algorithms for decremental APSP in expanders. However, the best such current algorithm only provides a rather weak tradeoff between the approximation factor and total update time: for bounded-degree expanders, the results of [CS21, Chu21] achieve $(\log n)^{O(1/\epsilon^2)}$-approximation with total update time $n^{1+O(\epsilon)}$, where ϵ is a given precision parameter.

In this paper, we propose to study a different kind of graphs, that we call *well-connected* graphs. Given an n-vertex graph G, a set S of its vertices called *supported vertices*, and parameters $\eta, d > 0$, we say that graph G is (η, d)-well-connected with respect to S, if, for every pair $A, B \subseteq S$ of disjoint equal-cardinality subsets of supported vertices, there is a collection $\mathcal{P}$ of paths in G, that connect every vertex of A to a distinct vertex of B, such that the length of each path in $\mathcal{P}$ is at most d, and every edge of G participates in at most η paths in $\mathcal{P}$. For intuition, it would be convenient to think of $d = 2^{\mathrm{poly}(1/\epsilon)}$, $\eta = n^{O(\epsilon)}$, and $|S| \geq |V(G)| - n^{1-\epsilon}$, for some parameter $0 < \epsilon < 1$. In the discussion below, we will informally refer to a graph G that is (η, d)-well-connected with respect to a set S of its vertices, with the above setting of parameters, as a *well-connected graph*. We develop an algorithmic toolkit for well-connected graphs that is, in a sense, analogous to some of the tools that are known for expander graphs. We then show that using well-connected graphs, together with these new algorithmic tools, allows us to overcome many of the hurdles mentioned above. For example, we obtain deterministic almost linear time $O(\mathrm{poly}\log n)$-approximation algorithms for Sparsest Cut, Lowest Conductance Cut, Minimum Balanced Cut, as well as a better deterministic algorithm for the Cut Player in the Cut-Matching Game, and better algorithm for Expander Decomposition and decremental

APSP in expanders. While our algorithmic toolkit immediately leads to strengthening the currently known algorithmic tools for expander graphs, it is our hope that they will eventually lead to replacing expanders with well-connected graphs in some of the applications mentioned above. We start by describing the algorithmic tools that we developed for well-connected graphs. We then provide an overview of several of applications of these results to static graph cut problems, and then discuss new results and potential future uses of these new algorithmic tools in dynamic APSP.

Before we continue, we need to introduce some notation. Given a graph G, a *cut* is a partition (A, B) of its vertices into non-empty subsets. The *sparsity* of the cut (A, B) is $\frac{|E(A,B)|}{\min\{|A|,|B|\}}$. Throughout this paper, we say that a graph G is a φ-expander, if every cut in G has sparsity at least φ. All graphs discussed in this paper are unweighted and undirected, unless stated otherwise. We will informally say that the running time or a total update time of an algorithm is *almost linear*, if it is bounded by $O(m^{1+o(1)})$, where m is the number of edges in the input graph (or the initial number of edges, if the graph is decremental). Given a graph G and a subset S of its vertices, the *volume* of S, denoted by $\mathrm{Vol}_G(S)$, is the sum of degrees of all vertices in S. The *volume of the graph* G, denoted by $\mathrm{Vol}(G)$, is $\mathrm{Vol}(G) = \mathrm{Vol}_G(V) = 2|E(G)|$. Given a collection M of pairs of vertices in graph G, and a collection $\mathcal{P}$ of paths in G, we say that the paths in $\mathcal{P}$ *route* the pairs in M if, for every pair $(u, v) \in M$ of vertices, there is a path in $\mathcal{P}$ whose endpoints are u and v. The *congestion* of a collection $\mathcal{P}$ of paths in a graph G is the largest number of paths that contain the same edge.

Algorithmic Tools for Well-Connected Graphs

The Distanced Matching Game. The first tool that we develop is a Distanced Matching Game, that can be viewed as an analogue of the Cut-Matching Game of [KRV09] for well-connected graphs. The input to the Distanced Matching Game consists of an integer n, a distance parameter d, and another parameter $0 < \delta < 1$. The game uses the notion of a *distancing*. Given an n-vertex graph G, a (δ, d)-distancing for G is a triple (A, B, E'), where A and B are disjoint equal-cardinality subsets of vertices of G with $|A| \geq n^{1-\delta}$, and E' is a subset of edges with $|E'| \leq |A|/16$. We require that the length of the shortest path connecting a vertex of A to a vertex of B in $G \setminus E'$ is at least d.

The Distanced Matching Game is played between a *Distancing Player* and a *Matching Player*. Similarly to the Cut-Matching Game, we start with a graph H that contains n vertices and no edges, and then iteratively add edges to H. In each iteration i, the Distancing Player needs to compute a (δ, d)-distancing (A_i, B_i, E'_i) in the current graph H, if it exists. The matching player then needs to compute an arbitrary matching $M_i \subseteq A_i \times B_i$, of cardinality at least $|A_i|/4$. The edges of M_i are added to graph H, and we continue to the next iteration. The game terminates when no (δ, d)-distancing exists in H (alternatively, we may terminate it earlier, if we establish that graph H has some desired properties, such as, e.g. it is well-connected).

Our first result shows that, if $d \geq 2^{4/\delta}$, then the number of iterations in the Distanced Matching Game is bounded by $n^{8\delta}$, regardless of the strategies of the two players. We note that this is significantly larger than the number of iterations in the Cut-Matching Game, which is typically bounded by $O(\mathrm{poly}\log n)$ (see, e.g. [KRV09, KKOV07]). However, as we show later, we gain an advantage that, under some specific strategy of the Distancing Player that we provide below, the resulting graph H is well-connected, and the distances between the vertices lying in the set S of supported vertices are quite low, as opposed to the (super)-logarithmic distances that expander graphs guarantee. We also note that, while it appears that the bound on the number of iterations is close to being tight, it is not clear whether the exponential dependence of the distance parameter d on $1/\delta$ in our result is necessary.

Hierarchical Support Structure and an algorithm for the Distancing Player. Our next result provides a deterministic algorithm for the Distancing Player. Suppose we are given a parameter

n, and a precision parameter ϵ. Let $\delta = 4\epsilon^3$ and $d = 2^{32/\epsilon^4}$, and let H be an n-vertex graph, that we can think of as arising during the execution of the Distanced Matching Game. The algorithm either computes a (δ, d)-distancing in graph H, or it computes a large set $S \subseteq V(H)$ of its vertices, such that H is $(\eta, \tilde{d})$-well-connected for S, where $\eta = n^{O(\epsilon)}$ and $\tilde{d} = 2^{O(1/\epsilon^5)}$. The running time of the algorithm is $O(|E(H)|^{1+O(\epsilon)})$.

In fact the above algorithm provides stronger guarantees. We define the notion of a Hierarchical Support Structure for an n-vertex graph H. Informally, the structure consists of a collection $\{H_1, \ldots, H_r\}$ of $r = \Omega(n^\epsilon)$ graphs, where $|V(H_i)| = \lceil n^{1-\epsilon} \rceil$ for all i, and an embedding of graph $\bigcup_{i=1}^{r} H_i$ into H via short paths that cause low congestion. For each one of the graphs H_i, we must in turn be given a Hierarchical Support Structure, that can be used to define a set $S(H_i)$ of supported vertices for H_i; graph H_i must be well-connected with respect to $S(H_i)$. We then let $S(H) = \bigcup_i S(H_i)$ be the set of supported vertices for graph H. We note that a somewhat similar notion was (implicitly) used in [CGL+20, CS21] in the context of expander graphs: that is, all graphs in the Hierarchical Support Structure were required to be expanders.

Our deterministic algorithm for the Distancing Player either provides the desired (δ, d)-distancing in graph H, or it constructs a Hierarchical Support Structure for H, so that H is well-connected with respect to the resulting set $S(H)$ of vertices. In all our subsequent results, we use the Distanced Matching Game with this implementation of the Distancing Player. This ensures that, once the algorithm terminates, we obtain a Hierarchical Support Structure for graph H, and a guarantee that H is well-connected with respect to $S(H)$.

APSP in Well-Connected Graphs. The Hierarchical Support Structure provides a convenient basis for decremental APSP in well-connected graphs obtained via the Distanced Matching Game. Indeed, suppose we are given such a graph H, that undergoes an online sequence of edge deletions. As edges are deleted from H, the graph may no longer be well-connected with respect to $S(H)$. We design a deterministic algorithm for decremental APSP that, given such a graph H, can withstand up to $|V(H)|^{1-\Theta(\epsilon)}$ edge deletions, as it maintains a set $S'(H) \subseteq S(H)$ of supported vertices. Over the course of the algorithm, vertices may leave $S'(H)$ but they may not join it, and $|S'(H)| \geq |V(H)|/2^{O(1/\epsilon)}$ holds at all times. The algorithm supports short-path queries between pairs of vertices in $S'(H)$: given a pair $x, y \in S'(H)$ of such vertices, it must return a path P of length at most $2^{O(1/\epsilon^6)}$ connecting them in the current graph H, in time $O(|E(P)|)$. The total update time of the algorithm is $O(|E(H)|^{1+O(\epsilon)})$. This algorithm can be thought of as mirroring similar algorithms for APSP in expanders of [CS21, Chu21] that we discuss below. For comparison, the best previous algorithm for APSP in bounded-degree expanders could only report paths whose length is bounded by $(\log n)^{O(1/\epsilon^2)}$ in response to short-path queries, with total update time is $O\left(n^{1+O(\epsilon)}\right)$, though it could withstand a longer sequence of edge deletions. Interestingly, the algorithmic tools presented so far can also be used to obtain better algorithms for APSP in expanders, as we discuss below. We note that it is currently not clear to us whether the exponential dependence of the lengths of the paths returned in response to short-path queries on $\mathrm{poly}(1/\epsilon)$ in our results is necessary. Improving this dependence may lead to further improvements in applications discussed below.

Applications to Other Algorithmic Tools

We use the machinery that we have developed in order to design better implementations of existing algorithmic tools. The first such tool that we discuss is the Cut-Matching Game of [KRV09].

Cut-Matching Game. The Cut-Matching Game was initially introduced by [KRV09], as part of their fast approximation algorithms for Minimum Balanced Cut and Sparsest Cut. The game has a single parameter n, that is an even integer, and it is played between two players: a Cut Player and a Matching

Player. The purpose of the game is to construct an n-vertex expander. The game starts with a graph H containing n vertices and no edges, and in every iteration edge are added to H, until we can certify that it becomes an expander. In the original version of the game suggested by [KRV09], in every iteration i, the cut player needs to compute a partition of $V(H)$ into two equal-cardinality subsets A_i and B_i, and the matching player needs to return an arbitrary perfect matching $M_i \subseteq A_i \times B_i$. The edges of M_i are then added to graph H, and the algorithm continues to the next iteration. In their paper, [KRV09] provided an algorithm for the Cut Player, that computes, in every iteration, a partition (A_i, B_i) of $V(H)$, so that, regardless of the strategy of the Matching Player, the algorithm is guaranteed to terminate with an $\Omega(1)$-expander after $O(\log^2 n)$ iterations. While the algorithm of [KRV09] for the Cut Player is very efficient – its running time is $\tilde{O}(n)$, it is unfortunately randomized, and it is unclear how to derandomize it, if our goal is an algorithm with almost linear running time.

Instead, we consider a variant of the Cut-Matching Game due to [KKOV07], that was also studied in [CGL$^+$20]. In this variant, in every iteration i, the Cut Player must either compute a partition (A'_i, B'_i) of $V(H)$ with $|A'_i| \geq |B'_i| \geq n/4$ and $|E_H(A'_i, B'_i)| \leq n/10$; or it must compute a subset $X \subseteq V(H)$ of at least $n/2$ vertices, so that graph $H[X]$ is a φ-expander (and we would like φ to be as close to 1 as possible). In the former case, the matching player must compute an arbitrary partition (A_i, B_i) of $V(H)$ with $|A_i| = |B_i|$ and $B'_i \subseteq B_i$, together with a perfect matching $M_i \subseteq A_i \times B_i$. The edges of M_i are added to H, and we continue to the next iteration. In the latter case, the matching player must compute an arbitrary matching $M_i \subseteq X \times V(H) \setminus X$, of cardinality $|V(H) \setminus X|$. The edges of M_i are then added to graph H, which is now guaranteed to be a $\varphi/2$-expander, and the algorithm terminates. As shown by [KKOV07], this variant of the game must terminate after $O(\log n)$ iterations[1]. Note that the question of obtaining a fast deterministic algorithm for the Cut Player in this variation of the game leads to a sort of chicken and egg situation: the Cut Player essentially needs to solve a Minimum Balanced Cut problem on the current graph H, and solving this problem efficiently typically requires running the Cut-Matching Game, which in turn requires an efficient implementation of the Cut Player.

In [CGL$^+$20], a deterministic algorithm for the Cut Player was presented for the above setting. For a given precision parameter $\epsilon \geq \frac{\log\log n}{(\log n)^{1/2}}$, the algorithm has running time $O\left(n^{1+O(\epsilon)}\right)$, and it ensures that the resulting graph has expansion $\varphi \geq 1/(\log n)^{1/\epsilon}$. The algorithm proceeds by constructing a hierarchical system of expanders that are embedded into H, similarly to our Hierarchical Support Structure, except that expanders are used instead of well-connected graphs. Unfortunately, it appears that the use of expanders forces one to lose a polylogarithmic in n factor in the expansion (or alternatively, in the length of the embedding paths) with every recursive level, which eventually leads to a rather weak expansion guarantee. In this paper we design a deterministic algorithm for the Cut Player with running time $n^{1+o(1)}$, that ensures that the resulting graph has expansion $1/(\log n)^{5+o(1)}$. Instead of using the approach of [CGL$^+$20], we rely on another algorithmic tool introduced in this paper, that we call *advanced path peeling*. We believe that this algorithmic tool is of independent interest, and in fact it can be used in order to either embed an expander graph into a given input graph G, or compute a sparse cut in G, thereby completely bypassing the Cut-Matching Game. Before we provide more details on advanced path peeling, we briefly mention a typical implementation of the Matching Player in the Cut-Matching Game, since it is related to the path peeling technique.

Path Peeling Algorithms. Typically, a Cut-Matching Game is used in order to either embed a large expander graph H into a given input graph G, or to compute a sparse cut in G. The game is played on graph H, that initially contains the set $V(G)$ of vertices and no edges. The Cut Player is implemented using one of the algorithms described above. In order to implement a Matching Player, in each iteration i, we try to construct a large large matching $M_i \subseteq A_i \times B_i$, and its routing $\mathcal{P}_i$ in

[1] In fact the game presented here is a slight modification of the game suggested by [KKOV07]. A formal proof that the number of iterations in this variations is still bounded by $O(\log n)$ appears in [CGL$^+$20].

G via short paths that cause low congestion. Matching M_i is then used in order to compute the response of the Matching Player. Paths in $\mathcal{P}_i$ also define an embedding of the edges of M_i into G, and so, as the algorithm progresses and new edges are added to H, we maintain an embedding of H into G. Once graph H becomes an expander, the algorithm terminates. Alternatively, if we are unable to compute a routing of a large matching $M_i \subseteq A_i \times B_i$, we would like to compute a sparse cut in G, and in this case the algorithm terminates with this cut. Therefore, in order to implement the Matching Player, we need an algorithm that, given a pair A_i, B_i of disjoint sets of vertices in a graph G, either computes a routing $\mathcal{P}_i$ of a large enough matching $M_i \subseteq A_i \times B_i$ via short paths that cause low congestion, or returns a sparse cut in G. In the original paper of [KRV09], the Matching Player was implemented via an algorithm for computing maximum flow and minimum cut. However, as was later observed in [CK19], it is sufficient to compute a *maximal* collection of such paths in G, by a simple greedy algorithm, that can be implemented very efficiently. We informally refer to such algorithms as *basic path peeling*. The algorithm keeps greedily adding short paths connecting vertices of A_i to vertices of B_i to set $\mathcal{P}_i$, while deleting from G edges that already participate in many paths. If, at the end of the algorithm, $|\mathcal{P}_i|$ is not sufficiently large, a simple application of the calssical Ball Growing technique is used to compute a sparse cut in G. As an example, in one implementation of such an algorithm (Theorem 3.2 of [CGL+20]), given a graph G, equal-cardinality sets A_i, B_i of vertices of G, and parameters $z > 0$ and $0 < \varphi < 1$, the algorithm either computes a cut (X, Y) of sparsity at most φ in G with $|X|, |Y| \geq z/2$; or it constructs a routing $\mathcal{P}_i$ of a matching $M_i \subseteq A_i \times B_i$, with $|M_i| \geq |A_i| - z$, such that paths in $\mathcal{P}_i$ have length at most $O\left(\frac{\Delta \log n}{\varphi}\right)$ each, and cause congestion at most $O\left(\frac{\Delta^2 \log^2 n}{\varphi^2}\right)$, where Δ is maximum vertex degree of G. The running time of the algorithm is $\tilde{O}(|E(G)|/\varphi^3)$. Using different methods (that rely on algorithms for approximate Maximum Flow), [CGL+20] design algorithms with better guarantees: the congestion is only bounded by $O(\Delta(\log n)/\varphi)$, and the running time is $m^{1+o(1)}$.

Consider now a more general setting (that we refer to as *advanced path peeling*), where we are given two sets A, B of vertices, and we need to route a *specific* matching $M \subseteq A \times B$. In this case, the input consists of an m-edge graph G, a collection $M = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of pairs of its vertices, and parameters $0 < \varphi < 1$ and $z > 0$. The goal is to either route a collection $M' \subseteq M$ of at least $k - z$ pairs of vertices via paths that are short and cause low congestion (compared to $1/\varphi$), or return a cut (X, Y) of sparsity at most φ in G, with $|X|, |Y| \geq \Omega(z)$. One could use the techniques employed in basic path peeling in order to design such an algorithm, but it seems inevitable that the running time of the algorithm can only be bounded by $O(mk)$.

In this paper, we provide a deterministic algorithm for advanced path peeling. For a given precision parameter ϵ, the algorithm has running time $O\left(\frac{m^{1+O(\epsilon)}}{\varphi^3}\right)$, and, in case a routing $\mathcal{P}$ is returned, the length of every path is bounded by $\frac{2^{O(1/\epsilon^6)} \cdot \Delta \cdot \log n}{\varphi}$, with congestion bounded by $\frac{2^{O(1/\epsilon^6)} \cdot \Delta^2 \cdot \log^2 n}{\varphi^2}$, where Δ is maximum vertex degree in G. The fact that we can pre-specify pairs of vertices to be routed makes advanced path peeling a much more powerful algorithmic tool than basic path peeling. For example, we can use it directly in order to either embed some fixed low-degree expander H into a given graph G, or to compute a sparse cut in G. This approach allows us to completely bypass the Cut-Matching Game, and we use it in some of our algorithms.

Applications to Static Graphs

Sparsest Cut and Lowest-Conductance Cut. Recall that the sparsity of a cut (A, B) in a graph G is $\frac{|E_G(A,B)|}{\min\{|A|,|B|\}}$. In the Sparsest Cut problem, the goal is to compute a cut of minimum sparsity in the input graph G. A closely related notion is that of *conductance*: the conductance of a cut (A, B) in G is

$\frac{|E_G(A,B)|}{\min\{\mathrm{Vol}_G(A),\mathrm{Vol}_G(B)\}}$. In the Lowest Conductance Cut problem, given a graph G, the goal is to compute a cut of smallest conductance. We define the *conductance* of a graph G, $\Psi(G)$, to be the smallest conductance value of any cut in G. Both Sparsest Cut and Lowest Conductance Cut problems are among the most fundamental optimization problems, and are routinely used in the design of graph algorithms, for example, when divide-and-conquer paradigm is involved. The best current approximation algorithms for these problems, due to [ARV09], achieve a factor-$O(\sqrt{\log n})$-approximation. Unfortunately, these algorithms are rather slow (though their running times are polynomially bounded), as they need to solve an SDP. The work of [KRV09], that introduced the Cut-Matching Game, provided a randomized $O(\log^2 n)$-approximation algorithm for both problems, with running time $\tilde{O}(m+n^{3/2})$. The super-linear running time in this algorithm is mostly due to the rather slow state of the art algorithms for (approximate) maximum flow that were available at the time. With the more recent improvements in such algorithms, the running time of the algorithm of [KRV09] becomes almost linear. Since the Sparsest Cut and Lowest Conductance Cut problems are so ubiquitous, it is however highly desirable to obtain fast *deterministic* algorithms for them. A number of deterministic algorithms, that are based on the Multiplicative Weights Update framework of [GK98, Fle00, Kar08], achieve a factor $O(\log n)$-approximation for both problems, in time $\tilde{O}(m^2)$. Additionally, several algorithms in which the approximation factor is roughly $O(\varphi^{1/2})$, where φ is the value of the optimal solution, are known (see e.g. [Alo86, ACL07, GLN$^+$19]; the algorithms achieve running times $\tilde{O}(n^\omega), \tilde{O}(mn)$, and $O(m^{1.5+0(1)})$, respectively). In [CGL$^+$20], deterministic algorithms for both Sparsest Cut and Lowest Conductance Cut problems were presented, that, for a parameter $\frac{\log\log n}{(\log n)^{1/2}} \leq \epsilon < 1$, achieve a factor $(\log n)^{O(1/\epsilon^2)}$-approximation in time $O(m^{1+\epsilon+o(1)})$. Unfortunately, in time $m^{1+o(1)}$, these algorithms cannot achieve a polylogarithmic approximation factor. Our improved algorithm for the Cut Player in the Cut-Matching Game immediately leads to deterministic algorithms for both Sparsest Cut and Lowest Conductance Cut problems, with approximation factor $O(\log^7 n \log\log n)$, and running time $m^{1+o(1)}$.

Minimum Balanced Cut. Minimum Balanced Cut is another classical graph partitioning problem that is extensively used in algorithm design. Given a graph G, we say that a cut (A,B) in G is β-balanced, if $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/\beta$. We say that the cut is *balanced*, if it is β-balanced for $\beta = 1/3$, and we say that it is *almost balanced*, if it is β-balanced for some absolute constant β. In the Minimum Balanced Cut problem, given a graph G, the goal is to compute a balanced cut (A,B) minimizing $|E_G(A,B)|$. It is quite common to use bicriteria approximation algorithms for the problem: a factor-α bicriteria algorithm must return an almost balanced cut (A,B) with $|E_G(A,B)| \leq \alpha \cdot \mathsf{OPT}$, where OPT is the lowest possible value of $|E_G(A',B')|$ for any balanced cut (A',B'). The seminal work of [ARV09] provides the best currently known bicriteria approximation algorithm for the problem, whose approximation factor is $O(\sqrt{\log n})$, though the algorithm is somewhat slow due to the need to solve an SDP. As with the Sparsest Cut and Lowest Conductance Cut problems, the randomized algorithm of [KRV09] can be used to obtain a factor-$O(\log^2 n)$ bicriteria approximation in time $O(m^{1+o(1)})$. The best current deterministic algorithm for the problem, due to [CGL$^+$20], obtained, for any $\frac{\log\log n}{(\log n)^{1/2}} \leq \epsilon < 1$, a factor-$(\log n)^{O(1/\epsilon^2)}$ bicriteria approximation, in time $O(m^{1+\epsilon+o(1)})$. As for Sparsest Cut and Lowest Conductance Cut problems, this algorithm can only achieve $(\log n)^{\omega(1)}$-approximation in time $m^{1+o(1)}$. In this paper we provide a deterministic bicriteria factor-$(\log n)^{8+o(1)}$ approximation algorithm, with running time $m^{1+o(1)}$.

We also consider an important variant of the problem, that we call Minimum Balanced Cut with Certificate. In this problem, we are given a graph G, and a target parameter ψ. The goal is to compute a cut (A,B) in G with $|E_G(A,B)| \leq \alpha\psi \cdot \mathrm{Vol}(G)$ (where α is the *approximation factor* of the algorithm), such that either $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/3$; or $\mathrm{Vol}_G(A) \geq 2\,\mathrm{Vol}(G)/3$, and graph $G[A]$ has conductance at least ψ. In the latter outcome, if $\mathrm{Vol}_G(B) \leq \mathrm{Vol}(G)/4$, we can view graph $G[A]$ as a

certificate that the value of the Minimum Balanced Cut in G is $\Omega(\psi \cdot |E(G)|)$. A factor-$\alpha$ approximation algorithm for Minimum Balanced Cut with Certificate can be easily converted into a factor-$O(\alpha)$ approximation algorithm for the MinimumBalancedCut problem, with running time that increases by at most factor $O(\log n)$ (this was shown in [CGL+20]; we also provide more details in Section 2.5). Algorithms for Minimum Balanced Cut with Certificate however appear to be significantly more powerful than those for Minimum Balanced Cut, as they can be used in order to compute expander decomposition of a given graph efficiently. In [CGL+20], a deterministic algorithm for Minimum Balanced Cut with Certificate that achieves approximation factor $\alpha = (\log n)^{O(1/\epsilon^2)}$, in time $O(m^{1+\epsilon+o(1)})$, for any $\frac{\log\log n}{(\log n)^{1/2}} \leq \epsilon < 1$, was presented. We provide a deterministic algorithm for Minimum Balanced Cut with Certificate with approximation factor $(\log n)^{8+o(1)}$, whose running time is $O(m^{1+o(1)}/\psi)$. For $\psi \geq 1/m^{o(1)}$, which is a common setting used in algorithms for expander decomposition, the running time becomes $O(m^{1+o(1)})$. We provide another algorithm, whose running time is $O(m^{1+o(1)})$ for any value of ψ and approximation factor remains the same, but it provides a somewhat weaker certificate: in case where $\mathrm{Vol}_G(B) < \mathrm{Vol}(G)/3$, it only guarantees that $G[A]$ contains a large subgraph with conductance at least ψ, but it does not compute such a graph.

Expander Decomposition. An (δ, ψ)-*expander decomposition* of a graph $G = (V, E)$ is a partition $\Pi = \{V_1, \ldots, V_k\}$ of the set V of vertices, such that for all $1 \leq i \leq k$, the conductance of graph $G[V_i]$ is at least ψ, and $\sum_{i=1}^{k} \delta_G(V_i) \leq \delta \cdot \mathrm{Vol}(G)$. Algorithms for expander decomposition are used extensively in the design of graph algorithms, in both static and dynamic settings. A long line of research [ST04, NS17, Wul17, SW19, ADK22] culminated in a randomized algorithm that computes a $(\delta, \delta/\operatorname{poly}(\log n))$-expander decomposition in time $\tilde{O}(m/\delta)$. The best previous deterministic algorithm, due to [CGL+20], computes a (δ, φ)-expander decomposition with $\varphi = \Omega(\delta/(\log m)^{O(1/\epsilon^2)})$, in time $O\left(m^{1+\epsilon+o(1)}\right)$. This algorithm was in turn used by [CGL+20] in order to obtain the first deterministic algorithms for Dynamic Connectivity and Dynamic Minimum Spanning Forest, with $n^{o(1)}$ worst-case update time, making a significant progress on a major open question in the area of dynamic algorithms. We provide a deterministic algorithm for computing a (δ, ψ)-expander decomposition of G with $\psi = \Omega\left(\frac{\delta}{(\log n)^{9+o(1)}}\right)$, in time $O(m^{1+o(1)}/\delta)$.

Applications to Dynamic Graphs: All-Pairs Shortest Paths (in Expanders)

In the decremental All-Pairs Shortest Paths (APSP) problem, the input is an n-vertex graph G with non-negative length on edges, that undergoes an online sequence of edge deletions. The goal is to support (approximate) shortest path query: given a pair x, y of vertices, return an (approximate) shortest path connecting x to y in the current graph G. We say that the algorithm achieves a factor-α approximation, if, in response to a shortest path query between x and y, it is guaranteed to return a path of length at most $\alpha \cdot \mathsf{dist}_G(x, y)$. Decremental, and, more generally, dynamic APSP is one of the most basic problem in the area of dynamic algorithms. It also has important connections to designing algorithms for classical cut and flow problems in the static graph model. For example, by combining the standard primal-dual technique-based algorithm of [GK98, Fle00] with an algorithm for a special case of decremental APSP, called Single-Source Shortest Paths (SSSP), one can obtain fast approximation algorithms for maximum s-t flow, minimum-cost s-t flow, minimum s-t cut, and so on, in both edge- and vertex-capacitated settings (see e.g. [CK19, Chu21]). Until recently, some of these algorithms provided the best available guarantees. Additionally, by combining the same techniques with the ideas of [Mad10] and the standard Ball Growing technique of [LR99, GVY95], we can essentially reduce the Maximum Multicommodity Flow and Minimum Multicut problems to decremental APSP. Indeed, a recent algorithm for APSP by [Chu21] has led to fast deterministic algorithms for Maximum Multicommodity Flow and Minimum Multicut: for any $\Theta(1/\log\log n) < \epsilon < 1$, the algo-

rithms achieve approximation factor $(\log m)^{2^{O(1/\epsilon)}}$, with running time $O\left(m^{1+O(\epsilon)}(\log m)^{2^{O(1/\epsilon)}}+k/\epsilon\right)$, where m is the number of edges in the input graph and k is the number of the demand pairs. Most likely any further improvements in the current guarantees for decremental APSP will immediately lead to improved bounds for both these problems. For comparison, the fastest previous approximation algorithms for Maximum Multicommodity Flow, achieving $(1+\epsilon)$-approximation, had running times $O(k^{O(1)}\cdot m^{4/3}/\epsilon^{O(1)})$ [KMP12] and $\widetilde{O}(mn/\epsilon^2)$ [Mad10], and we are not aware of any algorithms that achieve a faster running time with possibly worse approximation factors.

We now turn to discuss the APSP problem in more detail. In addition to the approximation factor that the algorithm achieves and its total update time (the time it takes to maintain its data structures), two other parameters of interest are query time (the time the algorithm takes to respond to shortest path query), and whether the algorithm can withstand an adaptive adversary. The latter means that the input sequence of edge deletions may depend on the responses to the queries that the algorithm returned so far, and even on the inner state of the algorithm. This is in contrast to the oblivious adversary setting, where the input sequence of edge deletions is fixed in advance. We note that a deterministic algorithm by definition can withstand an adaptive adversary. With four different parameters of interest to optimize, there is a vast amount of research achieving different tradeoffs between them; we will not attempt to present them all. Instead we will focus on a specific setting, where the algorithm must withstand an adaptive adversary (and is ideally deterministic), and the query time for shortest path query is bounded by $\tilde{O}(|E(P)|)$, where P is the path returned; note that this is close to the best possible query time. The main reason to restrict ourselves to these two constraints is that both these constraints are necessary in order to obtain almost-linear time algorithms for various cut- and flow problems in static graphs via methods mentioned above. Subject to these two constrains, we are interested in optimizing the tradeoff between the approximation factor that the algorithm achieves and its total update time. The best current algorithm for APSP in this setting, due to [Chu21], is a deterministic algorithm that achieves approximation factor $(\log m)^{2^{O(1/\epsilon)}}$, with total update time $O\left(m^{1+O(\epsilon)}\cdot(\log m)^{O(1/\epsilon^2)}\cdot\log L\right)$, for any $\Omega(1/\log\log m)\leq\epsilon<1$; here, L is the ratio of longest to shortest edge length. In contrast, the best current negative results only rule out obtaining a better than factor-4 approximation in time $O(n^{3-\delta})$ and query time $O(n^{1-\delta})$, for any constant $0<\delta<1$, under either the Boolean Matrix Multiplication, or the Online Boolean Matrix-Vertex Multiplication conjectures [DHZ00, HKNS15]. Furthermore, in the setting of oblivious adversary, a much better tradeoff between the approximation factor and total update time can be achieved: the algorithm of [Che18], for any $0<\epsilon<1$, obtains an $O(1/\epsilon)$-approximation, with total update time $O(m^{1+\epsilon+o(1)}\log L)$.

A very interesting special case of decremental APSP is decremental APSP on expanders. In this problem, the input graph G has unit edge lengths, and it is initially an expander. It is well known that, if an n-vertex graph G with maximum vertex degree Δ is a φ-expander, then for any pair x,y of its vertices, there is a path of length at most $O\left(\frac{\Delta\log n}{\varphi}\right)$ connecting them. Assume now that we are given an n-vertex φ-expander G as above, that undergoes a sequence of edge deletions. We would like to design an algorithm that supports short-path queries: given a pair x,y of vertices of G, return a path connecting x to y in the current graph G, whose length is at most $\alpha\cdot\frac{\Delta\log n}{\varphi}$, where we refer to α as the approximation factor of the algorithm[2]. As graph G undergoes edge deletion, it may no longer remain an expander, and distances between some of its vertices may grow significantly. In order to overcome this difficulty, we only ask that the algorithm maintains a large enough subset $S(G)\subseteq V(G)$ of *supported vertices*, and we restrict short-path queries to pairs of vertices in $S(G)$. We additionally

[2]Note that α is not necessarily an approximation factor strictly speaking, as it is possible that for a pair x,y of vertices queried, $\mathsf{dist}_G(x,y)\ll\frac{\Delta\log n}{\varphi}$. However, the approximation factors that we discuss are quite high, making this difference insignificant, and it is convenient for us to use the "approximation factor" notion for brevity.

require that the set $S(G)$ of supported vertices is decremental, that is, vertices may leave $S(G)$ over the course of the algorithm, but they may not join it. In its typical applications, the problem needs to be solved on expander graphs that arise from the Cut-Matching Game; these are usually expanders with maximum vertex degree $\Delta \leq O(\log^2 n)$ and expansion $\varphi = \Theta(1)$.

Decremental APSP in expanders is especially interesting for several reasons. First, it seems to be a relatively simple special case of APSP, and, if our goal is to obtain better algorithms for general APSP, solving the problem in expander graphs is a natural first step. As we discuss below, the bounds provided by the best current algorithms for APSP in expanders are nowhere near the best current negative bound of 4 on the approximation factor for APSP in general graphs. Second, this problem arises in various algorithms for *static* cut and flow problems, and seems to be intimately connected to efficient implementations of the Cut-Matching Game. Third, expander graphs are increasingly becoming a central tool for designing algorithms for various dynamic graph problems, and obtaining good algorithms for APSP in expanders will likely become a powerful tool in the toolkit of algorithmic techniques in this area. As such, we feel that it is crucial to obtain a good understanding of this problem.

The best previous algorithm for APSP in expanders due to [CS21] (see also [Chu21]), uses techniques similar to those in [CGL$^+$20], and provides the following guarantees[3]. Suppose we are given an n-vertex and m-edge graph G with maximum vertex degree Δ that is a φ-expander, which undergoes a sequence of at most $O\left(\frac{\varphi m}{\Delta}\right)$ edge deletions, and a precision parameter ϵ. The algorithm maintains a set U of vertices of G, that is incremental: that is, vertices may be added to U but not deleted from it. For every integer t, after t edges are deleted from G, we are guaranteed that $|U| \leq O(t\Delta/\varphi)$ holds. Throughout the algorithm, we let $S = V(G) \setminus U$ be the set of supported vertices. The algorithm supports short-path queries between pairs of vertices in S. Given such a pair $x, y \in S$, it returns a path P in $G[S]$ of length at most $O\left(\Delta^2(\log n)^{O(1/\epsilon^2)}/\varphi\right)$, with query time $O(|E(P)|)$. The total update time of the algorithm is $O\left(n^{1+O(\epsilon)}\Delta^7(\log n)^{O(1/\epsilon^2)}/\varphi^5\right)$. Assuming a typical setting where $\Delta \leq \operatorname{poly}\log n$, $\varphi \geq (1/\operatorname{poly}\log n)$, and $\epsilon \geq 1/(\log n)^{1/3}$, the algorithm achieves approximation factor $(\log n)^{O(1/\epsilon^2)}$, with total update time $O\left(n^{1+O(\epsilon)}\right)$. For example, if we wish to achieve a polylogarithmic approximation factor, then the total update time of the algorithm is only bounded by $O(n^{1+\delta})$ for some constant δ, and if we would like the total update time of the algorithm to be bounded by $n^{1+o(1)}$, then the approximation factor must be super-polylogarithmic, that is, $(\log n)^{\omega(1)}$.

We use the algorithmic tools that we develop for well-connected graphs in order to design a deterministic algorithm for APSP in expanders that achieves a better tradeoff between the approximation factor and total update time: our algorithm, when responding to short-path query between a pair $x, y \in S$ of vertices is guaranteed to return a path P connecting x to y, of length at most $\frac{2^{O(1/\epsilon^6)} \cdot \Delta^2 \cdot \log n}{\varphi}$, with query time $O(|E(P)|)$. The total update time of the algorithm is $O\left(\frac{m^{1+O(\epsilon)} \cdot \Delta^5}{\varphi^2}\right)$. If we consider again the setting where $\Delta \leq \operatorname{poly}\log n$ and $\varphi \geq 1/\operatorname{poly}\log n$, the algorithm achieves approximation factor $O(\operatorname{poly}\log n)$ with total update time $n^{1+o(1)}$. On the negative side, our algorithm can only withstand a somewhat shorter sequence of edge deletions: at most $O\left(n \cdot \varphi^2/\Delta^4\right)$ edges may be deleted, and it only guarantees that, after t edge deletions from G, $|U| \leq O\left(\Delta^4 t/\varphi^2\right)$. However, since the algorithm is typically used in the setting where $\Delta, 1/\varphi \leq O(\operatorname{poly}\log n)$, these drawbacks are generally insignificant. Another difference is that the path returned in response to a query by our algorithm is guaranteed to be contained in the current graph G, while the paths returned by the algorithm of [CS21, Chu21] is contained in $G \setminus U$. We are not aware of any negative implications of this difference.

[3]The algorithm from [CS21] was only analyzed for a specific setting of the parameters; a proof for the whole range of the parameters was provided in [Chu21].

Returning to the APSP problem in general graphs, the best current deterministic algorithm of [Chu21] uses APSP in expanders as its building block. As mentioned already, the algorithm of [Chu21] achieves approximation factor $(\log m)^{2^{O(1/\epsilon)}}$, with total update time $O\left(m^{1+O(\epsilon)} \cdot (\log m)^{O(1/\epsilon^2)} \cdot \log L\right)$. It seems conceivable that the techniques from [Chu21] can be used in order to improve the approximation factor to $(\log m)^{O(1/\operatorname{poly}(\epsilon))}$ with similar total update time, but there are significant obstacles to further improvements. The first such obstacle is that the algorithm relies on the best previous algorithm for APSP in expanders, in the setting where $\Delta \leq \operatorname{poly}\log n$ and $\varphi \geq 1/\operatorname{poly}\log n$, whose approximation factor is $(\log n)^{O(1/\epsilon^2)}$ with total update time $O\left(n^{1+O(\epsilon)}\right)$. Our new algorithm removes this obstacle. The second obstacle is that the algorithm from [Chu21] is recursive. The number of recursive levels is $O(1/\epsilon)$, and in each recursive level, a factor $(\log n)^{O(1/\epsilon^2)}$ is lost in approximation (due to the algorithm for APSP in expanders). Even when using our new algorithm for APSP in expanders, at least a $\operatorname{poly}\log n$ factor must be lost in each recursive level, resulting in a $(\log n)^{\Theta(1/\epsilon)}$ approximation factor. One of the main reasons for this polylogarithmic loss in each recursive level is that, when we rely on APSP in expanders, we are committing ourselves to at least a logarithmic loss in the approximation factor. The reason is that we typically only require that, in response to short-path query, the algorithm returns a path whose length is within factor α of $\frac{\Delta \log n}{\varphi}$, a quantity that bounds the diameter of the expander, even if the two queried vertices are very close to each other. Furthermore, one of the typical ways to exploit expanders is to first embed a large expander into the given input graph G, for example, using the Cut-Matching Game, and such an embedding typically does not preserve distances between vertices, except with a $\operatorname{poly}\log n$ distortion. We note that well-connected graphs do not suffer from this drawback, and it is our hope that they can replace expander graphs in future algorithms for APSP and related problems. With the Distanced Matching Game replacing the Cut-Matching Game, and our algorithm for APSP in well-connected graphs replacing APSP in expanders, it looks like we have a necessary toolkit in place for this. An improved algorithm for general APSP would likely immediately lead to improved approximation algorithms for Maximum Multicommodity Flow and Minimum Multicut via the techniques mentioned above.

Organization. For convenience, we provide a formal statement of our main results, and a high-level overview of our techniques in Section 2. We provide preliminaries in Section 3. In Section 4 we formally define the Distanced Matching Game, and prove the upper bound on its number of iterations. We formally define Hierarchical Support Structure in Section 5, and provide an algorithm for the Distancing Player in Section 6. We provide algorithms for decremental APSP in well-connected graphs and in expanders in Sections 7 and 8 respectively. Our algorithm for the Cut Player in the Cut-Matching Game is presented in Section 10. Finally, we present our algorithms for Sparsest Cut, Lowest Conductance Cut, Minimum Balanced Cut, and Expander Decomposition in Section 11.

2 Overview of Our Results and Techniques

2.1 The Distanced Matching Game and Related Algorithmic Toolkit

Let G be an n-vertex graph, and let $d > 0$ and $1 < \delta < 1$ be parameters. A (δ, d)-*distancing* in G is a triple (A, B, E'), where A, B are disjoint equal-cardinality subsets of vertices of G with $|A| \geq n^{1-\delta}$, and E' is a subset of edges of G with $|E'| \leq |A|/16$. We require that, in graph $G \setminus E'$, the smallest distance between a vertex of A and a vertex of B is at least d.

We introduce the Distanced Matching Game, that is played between two players: a Distancing Player and a Matching Player. The game can be thought of as an analogue of the Cut-Matching Game for well-connected graphs. The input to the game consists of an integral parameter n, and two additional parameters, $0 < \delta < 1$ and d. The game starts with a graph H containing n vertices and no edges, and

then proceeds in iterations. In every iteration some edges are inserted into H. In order to execute the ith iteration, the Distancing Player must provide a (δ, d)-distancing (A_i, B_i, E'_i) in the current graph H. The matching player must return a matching $M_i \subseteq A_i \times B_i$ of cardinality at least $|A_i|/8$. The matching cannot contain any pairs of vertices (x, y) for which an edge (x, y) lies in E'_i. We then add the edges of M_i to H, and continue to the next iteration. The game terminates when the distancing player can no longer compute a (δ, d)-distancing, though we may choose to terminate it earlier, if graph H has desired properties. Our first result bounds the number of iterations in the Distanced Matching Game:

Theorem 2.1 *Consider a* Distanced Matching Game *with parameters $n > 0, 0 < \delta < 1/4$ and d, such that $d \geq 2^{4/\delta}$ and $n^\delta \geq \frac{2^{14} \log n}{\delta^2}$. Then the number of iterations in the game is at most $n^{8\delta}$.*

This theorem can be thought of as an analogue of similar results of [KRV09, KKOV07], that bound the number of iterations in the Cut-Matching Game. The bound on the number of iteration that we obtain here is significantly higher that those for the Cut-Matching Game, which are typically bounded by $O(\text{poly} \log n)$. However, as we show below, we gain in other aspects – specifically, by constructing a large enough subset S of vertices of H, so that H is well-connected with respect to S. This in turn allows us to achieve significantly shorter distances between the vertices of S in H, than those guaranteed in expander graphs.

Our proof of Theorem 2.1 is completely different from the types of arguments that were used in order to bound the number of iterations in the Cut-Matching Game by [KRV09, KKOV07]. For all $i > 0$, let H_i denote graph H at the beginning of iteration i. Let $E' = \bigcup_i E'_i$, and, for all $i > 0$, let $H'_i = H_i \setminus E'$. We observe how the graphs $H'_1, H'_2, \ldots$ evolve over the course of the execution of the game (note that the set E' of edges is computed in hindsight, after the game terminates, so in a sense we "replay" the game to observe the evolution of these graphs). For all i, we define a partition $\mathcal{C}^i$ of the vertices of H_i into clusters. We ensure that the set $\mathcal{C}^{i+1}$ of clusters can only be obtained from set $\mathcal{C}^i$ by merging existing clusters. We say that a cluster $C \in \mathcal{C}^i$ belongs to level j, if $n^{\delta j} < |V(C)| \leq n^{\delta(j+1)}$. We also ensure that, for all j, the diameter of every level-j cluster in $\mathcal{C}_i$ is at most $2^{O(j)}$. If $C \in \mathcal{C}^i$ is a level-j cluster, then we say that all vertices of C lie at level j. If a vertex of H lies at level j of clustering $\mathcal{C}^i$, and at a level $j' > j$ of clustering $\mathcal{C}^{i+1}$, then we say that vertex v has been *promoted* over the course of iteration i. The key in the proof is to show that, once $\lceil n^{4\delta} \rceil$ iterations pass, a large number of vertices are promoted. Since every vertex may only be promoted at most $O(1/\delta)$ times, this is sufficient in order to bound the total number of iterations in the game.

Next, we define a Hierarchical Support Structure. The structure uses two main parameters: the base parameter $N > 0$, and a level parameter $j > 0$. We also assume that we are given a precision parameter $0 < \epsilon < 1$. The notion of Hierarchical Support Structure is defined inductively, using the level parameter j. If H is a graph containing N vertices, then a level-1 support structure for H simply consists of a set $S(H)$ of vertices of H, with $|V(H) \setminus S(H)| \leq N^{1-\epsilon^4}$. Assume now that we are given a graph H containing exactly N^j vertices. A level-j Hierarchical Support Structure for H consists of a collection $\mathcal{H} = \{H_1, \ldots, H_r\}$ of $r = N - \left\lceil 2N^{1-\epsilon^4} \right\rceil$ graphs, such that for all $1 \leq i \leq r$, $V(H_i) \subseteq V(H)$, and $V(H_1), \ldots, V(H_r)$ are all mutually disjoint. Additionally, it must contain, for all $1 \leq i \leq r$, a level-$(j-1)$ Hierarchical Support Structure for H_i, which in turn must define the set $S(H_i)$ of supported vertices for graph H_i. We require that each such graph H_i is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected with respect to $S(H_i)$, where $\tilde{d}_{j-1} = 2^{O(j/\epsilon^4)}$ and $\eta_{j-1} = N^{6+O(j\epsilon^2)}$. Lastly, the Hierarchical Support Structure for graph H must contain an embedding of graph $H' = \bigcup_{i=1}^r H_i$ into graph H, via path of length at most $2^{O(1/\epsilon^4)}$, that cause congestion at most $N^{O(\epsilon^2)}$. We then set $S(H) = \bigcup_{i=1}^r S(H_i)$, and we view $S(H)$ as the set of supported vertices for graph H, that is defined by the Hierarchical Support Structure.

We provide an algorithm for the Distancing Player of the Distanced Matching Game, that either produces the desired (δ, d)-distancing in the current graph H, or constructs a level-$\lceil 1/\epsilon \rceil$ Hierarchical Support Structure for H, together with a large set $S(H)$ of supported vertices, such that H is well-connected with respect to $S(H)$.

Theorem 2.2 *There is a deterministic algorithm, whose input consists of a parameter $0 < \epsilon < 1/4$, such that $1/\epsilon$ is an integer, an integer $N > 0$, and a graph H with $|V(H)| = N^{1/\epsilon}$, such that N is sufficiently large, so that $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^5}$ holds. The algorithm computes one of the following:*

- *either a (δ, d)-distancing (A, B, E') in H, where $\delta = 4\epsilon^3$, $d = 2^{32/\epsilon^4}$ and $|E'| \leq \frac{|A|}{N^{\epsilon^3}}$; or*
- *a level-$(1/\epsilon)$ Hierarchical Support Structure for H, such that graph H is $(\eta, \tilde{d})$-well-connected with respect to the set $S(H)$ of vertices defined by the support structure, where $\eta = N^{6+O(\epsilon)}$ and $\tilde{d} = 2^{O(1/\epsilon^5)}$.*

The running time of the algorithm is bounded by: $O(|E(H)|^{1+O(\epsilon)})$.

We note that our definition of the Hierarchical Support Structure ensures that $|S(H)| \geq |V(H)| \cdot \left(1 - \frac{1}{N^{\Omega(\epsilon^4)}}\right)$. The proof of Theorem 2.2 is similar to some of the arguments from [CGL+20], and arguments used in previous algorithms for decremental APSP in expanders by [CS21, Chu21]. We prove by induction on the level j that there is a deterministic algorithm, that, given a graph H with $|V(H)| = N^j$, either computes a (δ_j, d_j)-distancing in graph H (for appropriately chosen parameters δ_j and d_j), or computes a level-j Hierarchical Support Structure for H, such that H is $(\eta_j, \tilde{d}_j)$-well-connected with respect to the set $S(H)$ of vertices defined by the support structure, for appropriately chosen parameters $\eta_j, \tilde{d}_j$. The algorithm for level j proceeds as follows. We partition the vertices of H into N subsets $V_1, \ldots, V_N$, each containing N^{j-1} vertices. We then let $\mathcal{H}' = \{H_1, \ldots, H_N\}$ be an initial collection of graphs, where for all $1 \leq i \leq N$, $V(H_i) = V_i$ and $E(H_i) = \emptyset$. We run the Distanced Matching Game on all of the graphs of $\mathcal{H}'$ in parallel, with the level parameter $(j-1)$; the algorithm for the Distancing Player is obtained from the induction hypothesis for level $(j-1)$. The algorithm for the Matching Player performs a routing in graph H via basic path peeling, and is very similar to the algorithm employed together with Cut-Matching Game in numerous previous results, e.g. [CK19, CS21, CGL+20, Chu21]. If we successfully complete the Distanced Matching Game on at least $r' = \Omega(r)$ graphs of $\mathcal{H}$, that we denote by $\mathcal{H}' = \left\{H_{i_1}, \ldots, H_{i_{r'}}\right\}$, then we simultaneously obtain an embedding of graph $\bigcup_{z=1}^{r'} H_{i_z}$ into H, and also a guarantee that each graph $H_{i_z} \in \mathcal{H}'$ is well-connected with respect to the corresponding set $S(H_{i_z})$ of vertices that is defined by its Hierarchical Support Structure that the algorithm constructed . We then attempt to connect, for every pair $1 \leq z < z' \leq r'$ of indices, the sets $S(H_{i_z}), S(H_{i_{z'}})$ of vertices by many paths in graph H, so that the paths are sufficiently short and cause a low congestion. If we manage to do so for many such pairs z, z' of indices, then we will obtain a collection $\mathcal{H}'' \subseteq \mathcal{H}'$ of r graphs, and a certificate that graph H is well-connected with respect to the set $S(H) = \bigcup_{H_i \in \mathcal{H}''}^{r} S(H_i)$ of vertices. Otherwise, we will compute the required (δ_j, d_j)-distancing in graph H. Lastly, if we fail to complete the Distanced Matching Game on many of the graphs in $\mathcal{H}$, then we will also compute the required (δ_j, d_j)-distancing in graph H.

In all our subsequent algorithms, we will employ the Distanced Matching Game with the algorithm for the Distancing Player implemented by Theorem 2.2. Therefore, when the algorithm terminates, it outputs a level-$(1/\epsilon)$ Hierarchical Support Structure for the input graph H, together with a large set $S(H)$ of supported vertices, so that graph H is well-connected with respect to $S(H)$.

Lastly, we provide an algorithm for decremental APSP in a well-connected graph. Specifically, we assume that we are given a graph H that is an outcome with the Distanced Matching Game, in which

the Distancing Player is implemented by the algorithm from Theorem 2.2. Therefore, we are given a level-$(1/\epsilon)$ Hierarchical Support Structure for H, together with a large set $S(H)$ of its vertices, so that H is well-connected with respect to $S(H)$. We then assume that graph H undergoes a sequence of edge deletions. As edges are deleted from H, the well-connectedness property may no longer hold, and the Hierarchical Support Structure may be partially destroyed. Therefore, we only require that the algorithm maintains a large enough subset $S'(H) \subseteq S(H)$ of supported vertices, and that it can respond to short-path queries between pairs of vertices in $S'(H)$: given a pair x, y of such vertices, the algorithm needs to return a path of length at most $2^{O(1/\epsilon^6)}$ in the current graph H connecting them. We also require that the set $S'(H)$ is *decremental*, so vertices can leave this set but they may not join it. The result is summarized in the following theorem.

Theorem 2.3 *There is a deterministic algorithm, whose input consists of:*

- *a parameter $0 < \epsilon < 1/400$, so that $1/\epsilon$ is an integer;*
- *an integral parameter N that is sufficiently large, so that $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^6}$ holds;*
- *a graph H with $|V(H)| = N^{1/\epsilon}$; and*
- *a level-$(1/\epsilon)$ hierarchical support structure for H, such that H is $(\eta, \tilde{d})$-well-connected with respect to the set $S(H)$ of vertices defined by the Hierarchical Support Structure, where η and $\tilde{d}$ are parameters from Theorem 2.2.*

Further, we assume that graph H undergoes an online sequence of at most $\Lambda = |V(H)|^{1-10\epsilon}$ edge deletions. The algorithm maintains a set $S'(H) \subseteq S(H)$ of vertices of H, such that, at the beginning of the algorithm, $S'(H) = S(H)$, and over the course of the algorithm, vertices can leave $S'(H)$ but they may not join it. The algorithm ensures that $|S'(H)| \geq \frac{|V(H)|}{2^{4/\epsilon}}$ holds at all times, and it supports short-path queries between supported vertices: given a pair $x, y \in S'(H)$ of vertices, return a path P connecting x to y in the current graph H, whose length is at most $2^{O(1/\epsilon^6)}$, in time $O(|E(P)|)$. The total update time of the algorithm is $O(|E(H)|^{1+O(\epsilon)})$.

The algorithm for Theorem 2.3 is somewhat similar to the algorithm for APSP in expanders from [CS21]. Instead of proving Theorem 2.3 directly, we prove a more general theorem, that, for all $1 \leq j \leq 1/\epsilon$, given a graph H with $|V(H)| = N^j$ and a level-j Hierarchical Support Structure for H, such that H is well-connected with respect to the set $S(H)$ of vertices defined by the Hierarchical Support Structure, supports APSP in H, as the graph undergoes a limited number of edge deletions. The proof of the theorem is by induction on j. In order to obtain an algorithm for a fixed level j, we recursively maintain a data structure for APSP in graphs $H_1, \ldots, H_r \in \mathcal{H}$ that belong to the Hierarchical Support Structure of graph H. We also maintain, for all $1 \leq i \leq r$, an Even-Shiloach Tree data structure in graph H, that is rooted at the vertices of $S'(H_i)$. These data structures allow us to maintain a large enough decremental set $S'(H) \subseteq \bigcup_i S'(H_i)$ of vertices, and to support short-path queries between pairs of vertices in $S'(H)$ efficiently.

We compare this algorithm to the best previous algorithm for APSP in expanders, due to [CS21, Chu21]. For APSP in expanders, we consider a typical setting where the maximum vertex degree $\Delta = O(\operatorname{poly}\log n)$, and the expansion parameter is $\varphi = \Omega(1/\operatorname{poly}\log n)$, where n is the number of vertices in the input graph. For this setting, the algorithm of [CS21, Chu21] could only return paths between pairs of vertices from the supported set of length at most $(\log n)^{O(1/\epsilon^2)}$, compared to path length $2^{O(1/\epsilon^6)}$ of the above algorithm. The running time of both algorithms in this setting (assuming that ϵ is not too small) is similar. On the negative side, our algorithm can only withstand $n^{1-\Theta(\epsilon)}$

edge deletions, compared to the algorithm of [CS21], that can withstand up to $\Theta(m/\operatorname{poly}\log n)$ edge deletions. Also, the size $S(H)$ of supported vertices that the algorithm from [CS21] is significantly larger: it is $\Omega(n)$, compared to our bound of $n/2^{O(1/\epsilon)}$. Interestingly, the tools that we developed here allow us to obtain better algorithms for the APSP in expanders problem itself, as we show next.

2.2 Decremental APSP in Expanders

In the decremental APSP in expanders problem, the input is a graph G, that is initially φ-expander. The graph undergoes an online sequence of edge deletions. The algorithm needs to maintain a partition (S,U) of vertices of G into a set S of *supported* vertices, and a set U of *unsupported* vertices. As the algorithm progresses, vertices may be moved from S to U, but not in the opposite direction. The algorithm must support shorth-path query: given a pair $x,y\in S$ of supported vertices, return a short path P connecting x to y, in time $O(|E(P)|)$. Ideally, we would like to ensure that the algorithm can withstand a long enough sequence of edge deletions, and that the set S of supported vertices remains sufficiently large. We prove the following theorem for decremental APSP in expanders.

Theorem 2.4 *There is a deterministic algorithm, whose input consists of an n-vertex graph G with $|E(G)|=m$ that is a φ-expander for some $0<\varphi<1$, with maximum vertex degree at most Δ, and a parameter $\frac{2}{(\log n)^{1/12}}<\epsilon<\frac{1}{400}$, such that $1/\epsilon$ is an integer. We assume that graph G undergoes an online sequence of at most $\frac{n\cdot\varphi^2}{2^{13}\Delta^4}$ edge deletions. The algorithm maintains a set $U\subseteq V(G)$ of vertices, such that, for every integer $t>0$, after t edges are deleted from G, $|U|\leq\frac{2^{11}\Delta^4 t}{\varphi^2}$ holds. Vertex set U is incremental, so vertices may join it but they may not leave it. The algorithm also supports short-path query: given a pair of vertices $x,y\in V(G)\setminus U$, return an x-y path P in the current graph G, of length at most $\frac{2^{O(1/\epsilon^6)}\cdot\Delta^2\cdot\log n}{\varphi}$, with query time $O(|E(P)|)$. The total update time of the algorithm is $O\left(\frac{m^{1+O(\epsilon)}\cdot\Delta^5}{\varphi^2}\right)$.*

For a typical setting where $\Delta,1/\varphi=O(\operatorname{poly}\log n)$, the algorithm, in response to a short-path query, returns a path of length at most $2^{O(1/\epsilon^6)}\cdot\operatorname{poly}\log n$, with total update time $O(n^{1+O(\epsilon)})$. For the same setting, the best previous algorithm of [CS21], returned paths of length at most $(\log n)^{O(1/\epsilon^2)}$ in response to queries, and had similar total update time. On the negative side, the algorithm of [CS21, Chu21] could withstand a longer sequence of edge deletions, though in both cases it remains $\Omega(n/\operatorname{poly}\log n)$. The cardinality of the set U of unsupported vertices is somewhat lower in [CS21], though for this setting it remains in both cases $\Omega(t\cdot\operatorname{poly}\log n)$ after t edge deletions. Note that, for constant-degree expanders, by letting $\epsilon=(1/\log\log\log n)$, we can ensure that the paths returned in response to short-path queries have length at most $(\log n)^{1+o(1)}$, and the total update time of the algorithm is $n^{1+o(1)}$.

2.3 Advanced Path Peeling and Deterministic Algorithm for the Cut Player in the Cut-Matching Game

We prove the following theorem for advanced path peeling.

Theorem 2.5 *There is a deterministic algorithm, whose input consists of a connected n-vertex m-edge graph G, a collection $M=\{(s_1,t_1),\ldots,(s_k,t_k)\}$ of pairs of vertices in G, such that M is a matching, and parameters $0<\alpha\leq 1/2$, $0<\varphi<1$ and $\frac{4}{(\log n)^{1/24}}<\epsilon<\frac{1}{400}$. The algorithm computes one of the following:*

- *either a cut* (A,B) *with* $|E_G(A,B)| \leq \varphi \cdot \min\{|E_G(A)|, |E_G(B)|\}$*, and each of* A*,* B *contains at least* $\frac{\alpha k}{16}$ *vertices of set* $T = \{s_1, t_1, \ldots, s_k, t_k\}$*; or*
- *a routing* $\mathcal{P}$ *in* G *of a subset* $M' \subseteq M$ *containing at least* $(1-\alpha)k$ *pairs of vertices, such that every path in* $\mathcal{P}$ *has length at most* $\frac{2^{O(1/\epsilon^6)} \cdot \log n}{\varphi}$*, and the total congestion caused by the paths in* $\mathcal{P}$ *is at most* $\frac{2^{O(1/\epsilon^6)} \cdot \log n}{\varphi^2} \cdot \min\left\{\frac{1}{\alpha}, \log n\right\}$*.*

The running time of the algorithm is bounded by $O\left(\frac{m^{1+O(\epsilon)}}{\varphi^3}\right)$*.*

The idea in the proof of the theorem is to attempt to embed a well-connected graph H, whose vertex set is T, into G, via the Distanced Matching Game. We require that the embedding paths are short and cause low congestion. If we fail to do so, we will immediately obtain the desired sparse cut. Otherwise, we can rely on the algorithm for APSP in well-connected graphs from Theorem 2.3, together with an ES-Tree in graph G that is rooted at the set $S'(H)$ of supported vertices of H that the algorithm from Theorem 2.3 maintains, in order to support approximate shortest path query in graph G. We then greedily compute short paths routing pairs of vertices in M, while deleting edges that participate in too many paths from G. Once a large enough number of paths is routed (so the algorithm from Theorem 2.3 may no longer support short-path queries), we start the whole procedure from scratch.

Next, we provide the following deterministic algorithm for the Cut Player from the Cut-Matching Game.

Theorem 2.6 *There is a deterministic algorithm, that, given an* n*-vertex and* m*-edge graph* $G = (V,E)$ *with maximum vertex degree* Δ*, and a parameter* $\frac{2}{(\log n)^{1/25}} < \epsilon < \frac{1}{400}$*, returns one of the following:*

- *either a cut* (A,B) *in* G *with* $|A|, |B| \geq n/4$ *and* $|E_G(A,B)| \leq n/100$*; or*
- *a subset* $S \subseteq V$ *of at least* $n/2$ *vertices, such that graph* $G[S]$ *is* φ^**-expander, for* $\varphi^* \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \Delta^3 \cdot \log^2 n}\right)$*.*

The running time of the algorithm is $O\left(m^{1+O(\epsilon)} \cdot \Delta^7\right)$*.*

We note that, since the number of iterations in the Cut-Matching Game is bounded by $O(\log n)$, we can assume that $\Delta \leq O(\log n)$. By setting $\epsilon = 1/(\log\log\log n)^{1/6}$, we can then guarantee that $\varphi^* \geq \frac{1}{(\log n)^{5+o(1)}}$, and the running time of the algorithm is bounded by $O(n^{1+o(1)})$. In contrast, the algorithm of [CGL+20] could only achieve expansion $\varphi^* \geq 1/(\log n)^{1/\epsilon}$ with running time $n^{1+O(\epsilon)}$, and so in time $n^{1+o(1)}$ it could only achieve expansion $1/(\log n)^{\omega(1)}$.

Our techniques are different from those of [CGL+20], who rely on a recursive application of the Cut-Matching Game to smaller and smaller graphs. Instead, we compute a constant-degree n-vertex expander H, and then attempt to embed it into G using the algorithm for advanced path peeling from Theorem 2.5. If we successfully embed most edges of H into G, then, by invoking the expander pruning result of [SW19], we can compute a large enough subset $X \subseteq V(G)$ of vertices, such that $G[X]$ is a φ^*-expander. Otherwise, we obtain a sparse cut (A,B) in G with $|A| \geq |B|$. We then delete the vertices of B from G, and repeat this procedure. The algorithm continues as long as G contains at least $2n/3$ vertices. Once the number of vertices in G falls below $2n/3$, if we did not successfully embed an expander into G so far, then we obtain a sparse cut (A',B') in G, where A' contains all vertices that currently remain in G.

2.4 Sparsest Cut and Lowest Conductance Cut

We prove the following result for the Sparsest Cut and Lowest Conductance Cut problems.

Theorem 2.7 *There are deterministic algorithms for the* Sparsest Cut *and the* Lowest Conductance Cut *problems, that achieve a factor-$O(\log^7 n \log\log n)$-approximation in time $O\left(m^{1+o(1)}\right)$, where n and m are the number of vertices and edges, respectively, in the input graph.*

The best previous deterministic algorithm for both problems, due to [CGL+20], achieved a factor $(\log n)^{1/\epsilon^2}$-approximation, in time $O(m^{1+\epsilon})$, for any $\frac{\log\log n}{(\log n)^{1/2}} \leq \epsilon < 1$. Our algorithms for Sparsest Cut and Lowest Conductance Cut are essentially identical to those of [CGL+20], except that we plug in our stronger algorithm for the Cut Player in the Cut-Matching Game from Theorem 2.6 into their proof.

As in [CGL+20], we first consider the Most Balanced Sparse Cut problem. The input to the problem is an n-vertex graph G, and a parameter $0 < \varphi \leq 1$. The goal is to compute a cut (X, Y) in G of sparsity at most φ, while maximizing $\min\{|X|, |Y|\}$, that we refer to as the *size of the cut*. An (α, β)-bicriteria approximation algorithm for the problem, given parameters $0 < \varphi < 1$ and $z \geq 1$, must either compute a cut (X, Y) in G of sparsity at most φ and size at least z; or correctly establish that every cut (X', Y') whose sparsity is at most φ/α has size at most $\beta \cdot z$.

The problem is a natural intermediate step for obtaining fast algorithms for Sparsest Cut and Lowest Conductance Cut problems. It was first introduced independently by [NS17] and [Wul17], and has been studied extensively since (see e.g. [CK19, CS19, CGL+20]). As observed in previous work, a fast bicriteria approximation algorithm for this problem can be obtained by employing the Cut-Matching Game. In [CGL+20] (see Lemma 7.3), an (α, β)-bicriteria deterministic approximation algorithm was obtained for the Most Balanced Sparse Cut problem, with $\alpha = (\log n)^{O(1/\epsilon)}$ and $\beta = (\log n)^{O(1/\epsilon)}$, in time $O\left(m^{1+O(\epsilon)+o(1)} \cdot (\log n)^{O(1/\epsilon^2)}\right)$ for any $\frac{1}{c \log n} \leq \epsilon \leq 1$, for some fixed constant c. We obtain a deterministic (α, β)-bicriteria approximation algorithm with $\alpha = 2^{O(1/\epsilon^6)} \cdot \log^7 n$ and $\beta = 2^{O(1/\epsilon^6)} \cdot \log^6 n$, with running time $O\left(m^{1+O(\epsilon)+o(1)}\right)$, for any $\epsilon > 2/(\log n)^{1/24}$. For example, by setting $\epsilon = 1/(\log\log\log n)^{1/6}$, we can obtain an (α, β)-bicriteria approximation with $\alpha = (\log n)^{7+o(1)}$ and $\beta = (\log n)^{6+o(1)}$, and running time $m^{1+o(1)}$. In contrast, obtaining an (α, β)-bicriteria approximation with $\alpha = O(\log^c n)$ and $\beta = O(\log^c n)$ for any constant c, using the algorithm of [CGL+20] would result in a running time that can only be bounded by $m^{1+O(1/c)}$. On the other hand, if we restricted the running time to $m^{1+o(1)}$, then the results of [CGL+20] could not yield an (α, β)-bicriteria approximation in which both α and β are polylogarithmic in n. Our algorithm for the Most Balanced Sparse Cut is essentially identical to that of [CGL+20], except that it uses our stronger algorithm for the Cut Player in the Cut-Matching Game. Algorithms for Sparsest Cut and Lowest Conductance Cut easily follow from the algorithm for Most Balanced Sparse Cut, as shown in [CGL+20].

2.5 Minimum Balanced Cut and Expander Decomposition

We provide a deterministic factor-$(\log n)^{8+o(1)}$ approximation algorithm for Minimum Balanced Cut with Certificate problem, by proving the following theorem.

Theorem 2.8 *There is a deterministic algorithm, that, given a graph G with n vertices and m edges, and a parameter $\frac{\log^3 m}{m} < \psi \leq 1$, computes a cut (A, B) in G with $|E_G(A, B)| \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$, such that one of the following holds:*

- *either* $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/3$*; or*

- $\mathrm{Vol}_G(A) \geq 2\,\mathrm{Vol}(G)/3$, *and graph* $G[A]$ *has conductance at least* ψ.

The running time of the algorithm is $O(m^{1+o(1)}/\psi)$.

For $\psi \geq 1/m^{o(1)}$, which is a common setting used in algorithms for expander decomposition, our running time becomes $O(m^{1+o(1)})$. As mentioned already, [CGL+20] presented a deterministic algorithm for Minimum Balanced Cut with Certificate, that achieves approximation factor $\alpha = (\log n)^{O(1/\epsilon^2)}$, in time $O(m^{1+\epsilon})$, for any $\frac{\log\log n}{(\log n)^{1/2}} \leq \epsilon < 1$.

We provide another algorithm, that can be used in low-conductance regime, whose running time does not depend on ψ. Unfortunately, this algorithm provides a somewhat weaker certiciate if the cut that it returns is not balanced.

Theorem 2.9 *There is a deterministic algorithm, that, given a graph* G *with* n *vertices and* m *edges, and a parameter* $0 < \psi \leq 1$, *computes a cut* (A, B) *in* G *with* $|E_G(A,B)| \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$, *such that one of the following holds:*

- *either* $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/3$; *or*
- $\mathrm{Vol}_G(A) \geq 2\,\mathrm{Vol}(G)/3$, *and for every partition* (Z, Z') *of* A *with* $\mathrm{Vol}_G(Z), \mathrm{Vol}_G(Z') \geq \mathrm{Vol}(G)/100$, $|E_G(Z,Z')| \geq \psi \cdot \mathrm{Vol}(G)$.

The running time of the algorithm is $O(m^{1+o(1)})$.

The algorithm from Theorem 2.9 can be easily used to obtain a deterministic factor-$(\log n)^{8+o(1)}$ approximation algorithm for the Minimum Balanced Cut problem in time $O(m^{1+o(1)})$. Given an input graph G, we perform a binary search on the parameter ψ, until we find a value for which the algorithm from Theorem 2.9, when applied to G and ψ, returns a cut (A,B) with $|E_G(A,B)| \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$ and $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/4$; while, if applied to G and $\psi/2$, it returns a cut (A', B') with $|E_G(A',B')| \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$ and $\mathrm{Vol}_G(B') < \mathrm{Vol}(G)/4$. Note that (A,B) is an almost balanced cut, with $|E_G(A,B)| \leq \alpha\psi \cdot \mathrm{Vol}(G)$, where $\alpha = (\log n)^{8+o(1)}$. Let (A^*, B^*) be the optimal balanced cut, so $\mathrm{Vol}_G(A^*), \mathrm{Vol}_G(B^*) \geq \mathrm{Vol}(G)/3$. We claim that $|E_G(A^*,B^*)| \geq \frac{\psi}{2} \cdot \mathrm{Vol}(G)$. This is since cut (A^*, B^*) defines a partition of the set A' of vertices, that we denote by (Z, Z'), for which $\mathrm{Vol}_G(Z), \mathrm{Vol}_G(Z') \geq \mathrm{Vol}(G)/100$ must hold. Therefore, $|E_G(A^*,B^*)| \geq |E_G(Z,Z')| \geq \frac{\psi}{2} \cdot \mathrm{Vol}(G)$. We conclude that cut (A,B) is a factor-α bicriteria solution to instance G of Minimum Balanced Cut.

Our proofs of Theorem 2.8 and Theorem 2.9 depart from that of [CGL+20], who iteratively used the algorithm for Most Balanced Sparse Cut. The reason is that, while we obtain significantly better guarantees for the Most Balanced Sparse Cut problem, the approximation factor is still at least polylogarithmic in n, and, if we follow the framework of [CGL+20], who apply the algorithm for the Most Balanced Sparse Cut over the course of $O(1/\epsilon)$ iterations, we will still accumulate an approximation factor that is at least as high as $(\log n)^{\Theta(1/\epsilon)}$. Instead, we employ the Cut-Matching Game directly and iteratively. In every iteration, we either cut off a large enough subgraph of G via a low-conductance cut, or we (implicitly) embed a large expander into G.

Lastly, we consider expander decomposition. Recall that an (δ, ψ)-*expander decomposition* of a graph $G = (V, E)$ is a partition $\Pi = \{V_1, \ldots, V_k\}$ of the set V of vertices, such that for all $1 \leq i \leq k$, the conductance of graph $G[V_i]$ is at least ψ, and $\sum_{i=1}^{k} \delta_G(V_i) \leq \delta \cdot \mathrm{Vol}(G)$. We prove the following theorem.

Theorem 2.10 *There is a deterministic algorithm, that, given a graph G with n vertices and m edges, and a parameter $\frac{c\log^{13} m}{m} < \delta < 1$, where c is a large enough constant, computes a (δ, ψ)-expander decomposition of G with $\psi = \Omega\left(\frac{\delta}{(\log n)^{9+o(1)}}\right)$, in time $O(m^{1+o(1)}/\delta)$.*

The best previous deterministic algorithm, due to [CGL+20], computes a (δ, φ)-expander decomposition with $\varphi = \Omega(\delta/(\log m)^{O(1/\epsilon^2)})$, in time $O\left(m^{1+O(\epsilon)+o(1)}\right)$. Our algorithm is very similar to the algorithm of [CGL+20], except that we use the algorithm from Theorem 2.8 for the Minimum Balanced Cut problem, instead of its counterpart from [CGL+20].

3 Preliminaries

All logarithms in this paper are to the base of 2. All graphs are simple, undirected and unweighted, unless stated otherwise. Graphs with parallel edges are explicitly referred to as multigraph. Throughout the paper, we use a $\tilde{O}(\cdot)$ notation to hide multiplicative factors that are polynomial in $\log n$, where n is the number of vertices in the input graph.

We follow standard graph-theoretic notation. Given a graph G and two disjoint subsets A, B of its vertices, we denote by $E_G(A, B)$ the set of all edges with one endpoint in A and another in B, and by $E_G(A)$ the set of all edges with both endpoints in A. We also denote by $\delta_G(A)$ the set of all edges with exactly one endpoint in A. For a vertex $v \in V(G)$, we denote by $\delta_G(v)$ the set of all edges incident to v in G, and by $\deg_G(v)$ the degree of v in G. We may omit the subscript G when clear from context. Given a subset S of vertices of G, we denote by $G[S]$ the subgraph of G induced by S. We say that a subgraph C of G is a *cluster*, if C is a connected vertex-induced subgraph of G.

Matchings and Routings. If G is a graph, and $\mathcal{P}$ is a collection of paths in G, we say that the paths in $\mathcal{P}$ cause congestion η in G if every edge $e \in E(G)$ participates in at most η paths in $\mathcal{P}$, and some edge $e \in E(G)$ participates in exactly η such paths.

Let G be a graph, and let $M = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ be a collection of pairs of vertices of G. We say that M is a *matching* if every vertex $v \in V(G)$ participates in at most one pair in M, and for every pair $(s_i, t_i) \in M$, $s_i \neq t_i$. Note that we do not require that the pairs $(s_i, t_i) \in M$ correspond to edges of G. We say that a collection $\mathcal{P}$ of paths is a *routing* of the pairs in M in graph G, if $|\mathcal{P}| = k$, the paths in $\mathcal{P}$ are simple paths that are contained in G, and for every pair $(s_i, t_i) \in M$ of vertices, there is a path $P_i \in \mathcal{P}$ whose endpoints are s_i and t_i.

Assume now that we are given a graph G, two disjoint sets S, T of its vertices, and a collection $\mathcal{P}$ of paths. We say that the paths in $\mathcal{P}$ *route* vertices of S to vertices of T, or that $\mathcal{P}$ is a *routing* of S to T, if $\mathcal{P} = \{P(s) \mid s \in S\}$, and, for all $s \in S$, path $P(s)$ originates at vertex s and terminates at some vertex of T. We say that $\mathcal{P}$ is a *one-to-one routing* of S to T, if the endpoints of all paths in $\mathcal{P}$ are distinct.

Embeddings of Graphs. Let G, X be two graphs with $V(X) \subseteq V(G)$. An *embedding* of X into G is a collection $\mathcal{P} = \{P(e) \mid e \in E(X)\}$ of paths in graph G, such that, for every edge $e = (x, y) \in E(X)$, path $P(e)$ connects vertex x to vertex y. The *congestion* of the embedding is the maximum, over all edges $e' \in E(G)$, of the number of paths in $\mathcal{P}$ containing e'.

Given graphs G and X as above, and a subset $E' \subseteq E(X)$ of edges of X, an embedding of E' into G is defined similarly: it is simply an embedding of the subgraph of X induced by E'.

We will sometimes use a more general setting, where $V(X) \cap V(G) = \emptyset$. In this case, an embedding of X into G must include a mapping $\pi : V(X) \to V(G)$, where every vertex of X is mapped to a distinct vertex of G. Additionally, it must also include a collection $\mathcal{P} = \{P(e) \mid e \in E(X)\}$ of paths in graph G, such that, for every edge $e = (x, y) \in E(X)$, path $P(e)$ connects vertex $\pi(x)$ to vertex $\pi(y)$. The congestion of this embedding is defined as before.

We will use the following easy observation.

Observation 3.1 *There is a deterministic algorithm, whose input consists of a pair H, G of graphs with $V(H) \subseteq V(G)$, an embedding $\mathcal{P}$ of H into G, so that the paths in $\mathcal{P}$ have length at most d each and cause congestion at most η, a collection Π of pairs of vertices of H, and a collection $\mathcal{Q} = \{Q(u, v) \mid (u, v) \in \Pi\}$ of simple paths in H, such that, for every pair $(u, v) \in \Pi$ of vertices, path $Q(u, v)$ connects u to v, the paths in $\mathcal{Q}$ have length at most d' each, and cause congestion at most η' in H. The algorithm computes a collection $\mathcal{Q}' = \{Q'(u, v) \mid (u, v) \in \Pi\}$ of paths in graph G, such that, for every pair $(u, v) \in \Pi$ of vertices, path $Q'(u, v)$ connects u to v, the paths in $\mathcal{Q}'$ have length at most $d \cdot d'$ each, and cause congestion at most $\eta \cdot \eta'$ in G. The running time of the algorithm is at most $O\left(\min\left\{|\Pi| \cdot d \cdot d', |E(G)| \cdot \eta \cdot \eta'\right\}\right)$.*

Proof: We process every pair $(u, v) \in \Pi$ one by one. When pair (u, v) is processed, we consider the path $Q(u, v) \in \mathcal{Q}$, and we denote the sequence of edges on $Q(u, v)$ by $(e_1, e_2, \ldots, e_r)$, where $r \leq d'$. For all $1 \leq i \leq r$, let $P(e_i) \in \mathcal{P}$ be the path that serves as the embedding of edge e_i, whose length must be at most d. We obtain path $Q'(u, v)$ connecting u to v in G by concatenating the paths $P(e_1), P(e_2), \ldots, P(e_r)$. It is immediate to verify that the length of path $Q'(u, v)$ is at most $d \cdot d'$. Let $\mathcal{Q}' = \{Q'(u, v) \mid (u, v) \in \Pi\}$ be the resulting set of paths. Consider any edge $e \in E(G)$, and let $S(e)$ be the collection of all edges $e' \in E(H)$ with $e \in P(e')$, where $P(e') \in \mathcal{P}$ is the embedding path of e'. Then $|S(e)| \leq \eta$, and every edge $e' \in S(e)$ participates in at most η' paths in $\mathcal{Q}$. Therefore, edge e may participate in at most $\eta \cdot \eta'$ paths in $\mathcal{Q}'$, and so the congestion that the paths in $\mathcal{Q}'$ cause in G is at most $\eta \cdot \eta'$. It is immediate to verify that every pair $(u, v) \in P$ of vertices can be processed in time $O(d \cdot d')$, and so the total running time of the algorithm is at most $O(|\Pi| \cdot d \cdot d')$. Since every edge of $E(G)$ belongs to at most $\eta \cdot \eta'$ paths in $\mathcal{Q}'$, it is also easy to verify that the running time is bounded by $O(\eta \cdot \eta' \cdot |E(G)|)$. □

Distances and Balls. Given a graph G, for a pair $u, v \in V(G)$ of its vertices, we denote by $\mathsf{dist}_G(u, v)$ the *distance* between u and v in G, that is, the length of the shortest path between u and v. For a pair S, T of subsets of vertices of G, we define the distance between S and T to be $\mathsf{dist}_G(S, T) = \min_{s \in S, t \in T} \mathsf{dist}_G(s, t)$. For a vertex $v \in V(G)$, and a vertex subset $S \subseteq V(G)$, we also define the distance between v and S as $\mathsf{dist}_G(v, S) = \min_{u \in S} \mathsf{dist}_G(v, u)$. The *diameter* of the graph G, denoted by $\mathsf{diam}(G)$, is the maximum distance between any pair of vertices in G. For a vertex $v \in V(G)$ and a distance parameter $D \geq 0$, we denote by $B_G(v, D) = \{u \in V(G) \mid \mathsf{dist}_G(u, v) \leq D\}$ the *ball of radius D around v*. Similarly, for a subset $S \subseteq V(G)$ of vertices, we let the ball of radius D around S be $B_G(S, D) = \{u \in V(G) \mid \mathsf{dist}_G(u, S) \leq D\}$. We will sometimes omit the subscript G when clear from context.

3.1 Dynamic Algorithms

Dynamic Graphs. Consider a graph G that undergoes an online sequence $\Sigma = (\sigma_1, \sigma_2, \ldots)$ of edge deletions, that we may also refer to as *updates*. After each update operation, the algorithm will perform some updates to the data structures that it maintains. We refer to different "times" during the algorithm's execution. The algorithm starts at time 0. For each $t \geq 0$, we refer to "time t in the

algorithm's execution" as the time immediately after all updates to the data structures maintained by the algorithm following the tth edge deletion $\sigma_t \in \Sigma$ are completed. When we say that some property holds at every time during the algorithm's execution, we mean that the property holds at all times t of the algorithm's execution, but it may not hold, for example, during the procedure that updates the data structures maintained by the algorithm, following some edge deletion $\sigma_t \in \Sigma$. For $t \geq 0$, we denote by $G^{(t)}$ the graph G at time t; that is, $G^{(t)}$ is the original graph, and for $t \geq 0$, $G^{(t)}$ is the graph obtained from G after the first t edge deletions $\sigma_1, \ldots, \sigma_t$.

We say that a set S of elements is *decremental* if, over time, elements can be deleted from S but they may not be added to S. Similarly, we say that S is *incremental* if elements can be added to S as the time progresses, but not deleted from S.

Even-Shiloach Trees [ES81, Din06, HK95]. Suppose we are given a graph $G = (V, E)$ with integral lengths $\ell(e) \geq 1$ on its edges $e \in E$, a source vertex s, and a distance bound $D \geq 1$. Even-Shiloach Tree (ES-Tree) algorithm maintains, for every vertex v with $\mathsf{dist}_G(s, v) \leq D$, the distance $\mathsf{dist}_G(s, v)$, under the deletion of edges from G. Moreover, it maintains a shortest-path tree τ rooted at vertex s, that includes all vertices v with $\mathsf{dist}_G(s, v) \leq D$. We denote the corresponding data structure by ES-Tree(G, s, D), or just ES-Tree when clear from context. The total running time of the algorithm, including the initialization and all edge deletions, is $O(m \cdot D \log n)$, where m is the initial number of edges in G and $n = |V|$.

3.2 Cuts, Flows, Sparsity, Conductance and Expanders.

Even though all graphs that we deal with are undirected, it will sometimes be useful to assign directions to paths in such a graph. In order to do so, for a path P in an undirected graph G, we designate one of its endpoints (say u) as the *first endpoint* of P, and the other endpoint (say v) as its *last endpoint*. We may then say that path P is *directed from u to v*, or that it originates at u and terminates at v. If $\mathcal{P}$ is a collection of path in an undirected graph G, and we have assigned a direction to each of the paths, we may refer to $\mathcal{P}$ as a *collection of directed paths*, even though graph G is undirected.

Flows. Let G be a graph, and let $\mathcal{P}$ be a collection of directed paths in graph G. A *flow* over the set $\mathcal{P}$ of paths is an assignment of non-negative values $f(P) \geq 0$, called *flow-values*, to every path $P \in \mathcal{P}$. We sometimes refer to paths in $\mathcal{P}$ as *flow-paths for flow f*. For each edge $e \in E(G)$, let $\mathcal{P}(e) \subseteq \mathcal{P}$ be the set of all paths whose first edge is e, and let $\mathcal{P}'(e) \subseteq \mathcal{P}$ be the set of all paths whose last edge is e. We say that edge e *sends z flow units* in f if $\sum_{P \in \mathcal{P}(e)} f(e) = z$, and we say that edge e *receives z flow units* in f if $\sum_{P \in \mathcal{P}'(e)} f(P) = z$. Similarly, for a vertex $v \in V(G)$, we say that v sends z flow units in f if the sum of flow-values of all paths $P \in \mathcal{P}$ that originate at v is z. We say that v receives z flow units in f if the sum of the flow-values of all paths $P \in \mathcal{P}$ that terminate at v is z. The *congestion* that flow f causes on an edge e is $\sum_{\substack{P \in \mathcal{P}: \\ e \in E(P)}} f(P)$, and the *total congestion* of the flow f is the maximum congestion that it causes on any edge $e \in E(G)$.

Cuts and Expansion. Given a graph $G = (V, E)$, a *cut* in G is a bipartition (A, B) of the set V of its vertices, with $A, B \neq \emptyset$. The *sparsity* of the cut (A, B) is $\varphi_G(A, B) = \frac{|E_G(A,B)|}{\min\{|A|,|B|\}}$. We denote by $\Phi(G)$ the smallest sparsity of any cut in G, and we refer to $\Phi(G)$ as the *expansion* of G.

Expanders. We define the notion of expanders using graph expansion.

Definition 3.1 (Expander) *We say that a graph G is a φ-expander, for a parameter $0 < \varphi < 1$, if $\Phi(G) \geq \varphi$.*

We will sometimes informally say that graph G is an *expander* if $\Phi(G)$ is a constant independent of $|V(G)|$. We use the following immediate observation, that was also used in previous works, (see e.g. Observation 2.3 in [CGL+20]).

Observation 3.2 *Let $G = (V, E)$ be an n-vertex graph that is a φ-expander, and let G' be another graph that is obtained from G by adding to it a new set V' of at most n vertices, and a matching M, connecting every vertex of V' to a distinct vertex of G. Then G' is a $\varphi/2$-expander.*

We also use the following theorem that provides a fast algorithm for an explicit construction of an expander, that is based on the results of Margulis [Mar73] and Gabber and Galil [GG81]. The proof was shown in [CGL+20].

Theorem 3.3 (Theorem 2.4 in [CGL+20]) *There is a constant $\alpha_0 > 0$ and a deterministic algorithm, that we call* ConstructExpander*, that, given an integer $n > 1$, in time $O(n)$ constructs a graph H_n with $|V(H_n)| = n$, such that H_n is an α_0-expander, and every vertex in H_n has degree at most 9.*

Expander Pruning. We use an algorithm for expander pruning by [SW19]. We slightly rephrase it so it is defined in terms of graph expansion, instead of conductance that was used in the original paper. This variation of the original expander pruning theorem of [SW19] was proved explicitly in [Chu21] (see Theorem 2.2 in full version of the paper).

Theorem 3.4 (Adaptation of Theorem 1.3 in [SW19]; see Theorem 2.2 in [Chu21]) *There is a deterministic algorithm, that, given an access to the adjacency list of a graph G that is a φ-expander, for some parameter $0 < \varphi < 1$, such that the maximum vertex degree in G is at most Δ, and a sequence $\Sigma = (e_1, e_2, \ldots, e_k)$ of $k \leq \frac{\varphi |E(G)|}{10\Delta}$ online edge deletions from G, maintains a set $\tilde{U} \subseteq V(G)$ of vertices, with the following properties. Let $G^{(i)}$ denote the graph $G \setminus \{e_1, \ldots, e_i\}$; let $\tilde{U}_0 = \emptyset$ be the set $\tilde{U}$ at the beginning of the algorithm, and for all $0 < i \leq k$, let $\tilde{U}_i$ be the set $\tilde{U}$ after the deletion of the edges of $e_1, \ldots, e_i$ from graph G. Then, for all $1 \leq i \leq k$: $\tilde{U}_{i-1} \subseteq \tilde{U}_i$; $|\tilde{U}_i| \leq \frac{8i\Delta}{\varphi}$; and graph $G^{(i)} \setminus \tilde{U}_i$ is a $\frac{\varphi}{6\Delta}$-expander. The total running time of the algorithm is $\widetilde{O}(k\Delta^2/\varphi^2)$.*

Graph Conductance. For a graph $G = (V, E)$ and a subset $S \subseteq V$ of its vertices, the *volume* of S is $\mathrm{Vol}_G(S) = \sum_{v \in S} \deg_G(v)$. We denote by $\mathrm{Vol}(G) = \mathrm{Vol}_G(V)$. The *conductance* of a cut (A, B) in G is: $\psi_G(A, B) = \frac{|E_G(A,B)|}{\min\{\mathrm{Vol}_G(A), \mathrm{Vol}_G(B)\}}$. We denote by $\Psi(G)$ the smallest conductance of any cut in G, and we refer to $\Psi(G)$ as the *conductance* of G.

3.3 Embeddings with Fake Edges and Expansion

Typically, when using the Cut-Matching game, we either embed an expander graph H with $V(H) = V(G)$ into the given graph G, or compute a sparse cut (A, B) in G. Unfortunately, it is possible that one side of the cut, say A, is quite small in the latter case. This often poses challenges in applications of the Cut-Matching game where the goal is to obtain very efficient algorithms. This is since we essentially spend time $\Omega(|E(G)|)$ in order to execute the Cut-Matching game, and end up computing a sparse cut whose one side may be very small. Ideally, for efficient algorithms, it is desirable that the sparse cut that we compute is as balanced as possible. A standard way to overcome this issue, that

was suggested in the original paper of [KRV09] that introduced the Cut-Matching game, is to use *fake edges*. Intuitively, we will augment the graph G with a small number of edges, that we refer to as fake edges, to indicate that they do not actually lie in G. If F is the set of fake edges, we will denote by $G+F$ the graph obtained by adding the edges of F into G. We will use the Cut-Matching game to either compute a sparse cut in G, whose both sides are relatively large; or to compute an embedding of some expander graph H into $G+F$. In the latter case, both the embedding and the set F of fake edges are constructed during the Cut-Matching game, and we will then extract a large expander graph from G. The following lemma from [CGL+20] provides an algorithm to extract a large expander graph from G efficiently.

Lemma 3.5 (Lemma 2.9 from [CGL+20]) *Let G be an n-vertex graph, and let H be another graph with $V(H)=V(G)$, with maximum vertex degree Δ_H, such that H is a ψ-expander, for some $0<\psi<1$. Let F be any set of k fake edges for G, and let Δ_G be the maximum vertex degree in $G+F$. Assume that there exists an embedding $\mathcal{P}=\{P(e) \mid e\in E(H)\}$ of H into $G+F$, that causes congestion at most η, for some $\eta\geq 1$. Assume further that $k\leq \frac{\psi n}{32\Delta_G\eta}$. Then there is a subgraph $G'\subseteq G$ that is a ψ'-expander, for $\psi'\geq \frac{\psi}{6\Delta_G\cdot\eta}$, such that, if we denote by $A=V(G')$ and $B=V(G)\setminus A$, then $|A|\geq n-\frac{4k\eta}{\psi}$ and $|E_G(A,B)|\leq 4k$. Moreover, there is a deterministic algorithm, that we call* AlgExtractExpander*, that, given $G,H,\mathcal{P}$ and F, computes such a graph G' in time $\tilde{O}(|E(G)|\Delta_G\cdot\eta/\psi)$.*

3.4 The Cut-Matching Game

The Cut-Matching Game was introduced by Khandekar, Rao, and Vazirani [KRV09] as a tool for obtaining fast approximation algorithms for the Sparsest Cut and Minimum Balanced Cut problems. We describe here a variant of this game, that was introduced by Khandekar et al. [KKOV07], and later slightly modified by [CGL+20]. The game is played between two players, the *Cut Player*, and the *Matching Player*. The game uses a parameter n, which is an even integer. The purpose of the game is to construct an n-vertex expander graph H. At the beginning of the game, graph H contains n vertices and no edges, and then in every iteration some edges are added to H. For intuition, it may be convenient to think of the Cut Player's goal being to construct the expander in as few iterations as possible, and the Matching Player's goal as trying to delay the construction of the expander.

The game starts with graph H containing n vertices and no edges. The ith iteration is played as follows. The Cut Player either computes a partition (A_i,B_i) of $V(H)$ with $|A_i|,|B_i|\geq n/4$ and $|E_H(A_i,B_i)|\leq n/100$; or it computes a set $X\subseteq V(H)$ with $|X|\geq n/2$, such that $H[X]$ is a φ-expander, for some expansion parameter $0<\varphi<1$. Assume first that the former happens, and assume without loss of generality that $|A_i|\leq|B_i|$. The Matching Player must compute any partition (A'_i,B'_i) of $V(H)$ with $|A'_i|=|B'_i|$, such that $A_i\subseteq A'_i$, and then it must compute an arbitrary perfect matching M_i between A'_i and B'_i. The edges of M_i are then added to the graph H, and the algorithm continues to the next iteration. If the latter case happens, that is, the Cut Player returns a set $X\subseteq V(H)$ of at least $n/2$ vertices, so that $H[X]$ is a φ-expander, denote $Y=V(H)\setminus X$. The Matching Player must then compute a matching $M_i\subseteq X\times Y$ with $|M_i|=Y$. The edges of M_i are added to graph H, and the algorithm terminates. In this case, from Observation 3.2, we are guaranteed that H is a $\varphi/2$-expander. The following theorem follows directly from the result of [KKOV07], and was proved explicitly in [CGL+20] (see Theorem 2.5 in the full version).

Theorem 3.6 *There is a constant c, such that the algorithm described above terminates after at most $c\log n$ iterations.*

3.5 Graph Cutting and Partitioning

We use several graph cutting and partitioning procedures, that exploit standard tools. We start with procedure ProcCut, which is a variation of Leighton and Rao's ball growing technique [LR99]. In all these procedures, the input is a graph G, with a subset T of vertices of G called terminals. The goal is to either compute a single cluster, or a collection of clusters in G with some specific properties. Throughout, a subgraph $C \subseteq G$, we denote by $T_C = T \cap V(C)$ the set of all terminals contained in C.

3.5.1 Procedure ProcCut

The input to the procedure is an n-vertex graph G, a set $T \subseteq V(G)$ of k vertices called terminals, a specific terminal $t_C \in T$, and distance parameters d and Δ.

The procedure returns a cluster $C \subseteq G$, and a subset $\hat{T}_C \subseteq T$ of terminals, for which the following properties hold:

C1. $T_C \subseteq \hat{T}_C$;

C2. $|\hat{T}_C| \leq |T_C| \cdot k^{64/\Delta}$;

C3. $V(C) \subseteq B_G(t_C, \Delta \cdot d)$;

C4. $\hat{T}_C \subseteq B_G(t_C, \Delta \cdot d)$; and

C5. for every pair $x \in V(C)$, $t' \in T \setminus \hat{T}_C$ of vertices, $\mathsf{dist}_G(v, t') \geq 4d$.

The following lemma summarizes Procedure ProcCut.

Lemma 3.7 *There is a deterministic algorithm called* ProcCut*, that, given an n-vertex graph G, a subset $T \subseteq V(G)$ of k vertices called terminals, a specific terminal $t_C \in T$, and parameters $d, \Delta > 0$, computes a cluster $C \subseteq G$ together with a set $\hat{T}_C \subseteq T$ of terminals, for which properties (C1) – (C5) hold. The running time of the algorithm is $O(|E(C)| \cdot n^{64/\Delta})$.*

Proof: We assume that we are given as input an n-vertex graph G, a set $T \subseteq V(G)$ of k vertices called terminals, together with a specific terminal $t_C \in T$ that we denote by t for simplicity, and distance parameters d and Δ. The procedure performs a breadth-first-search (BFS) from vertex t in graph G, up to a certain depth, that will be determined later.

For all $i \geq 1$, we denote by L_i the set of all vertices of G that lie at distance $4(i-1)d+1$ to $4id$ from t in G. In other words:

$$L_i = B_G(t, 4id) \setminus B_G(t, 4(i-1)d).$$

We refer to the vertices of L_i as *layer i of the BFS*. We denote by k_i the number of terminals lying in $L_1 \cup \cdots \cup L_i$. We also denote by m_i the total number of edges of G whose both endpoints lie in $L_1 \cup \cdots \cup L_i$. The following definition is crucial for the description of Procedure ProcCut.

Definition 3.2 (Eligible Layer) *For an integer $i > 1$, we say that layer L_i of the BFS is* eligible *if both of the following two conditions hold:*

L1. $m_i \leq m_{i-1} \cdot n^{64/\Delta}$*; and*

L2. $k_i \leq k_{i-1} \cdot k^{64/\Delta}$

We need the following claim, whose proof uses standard arguments.

Claim 3.8 *There exists an index* $1 < i \leq \Delta/8$*, such that layer* L_i *is eligible.*

Proof: Assume otherwise. Then for all $1 < i \leq \Delta/8$, layer L_i is ineligible. For each such index i, we say that layer L_i is *type-1 ineligible* if it violates Condition (L1). Otherwise, we say that it is *type-2 ineligible*, in which case it must violate Condition (L2). Since every layer L_i with $1 < i \leq \Delta/8$ is ineligible, either there are at least $\Delta/32$ type-1 ineligible layers L_i with $1 < i \leq \Delta/8$, or there are at least $\Delta/32$ type-2 ineligible layers L_i with $1 < i \leq \Delta/8$. We now consider each of the two cases and prove that they are impossible.

Assume first that there are at least $\Delta/32$ type-1 ineligible layers L_i with $1 < i \leq \Delta/8$, and denote their indices by $i_1, i_2, \ldots, i_z$, where $1 < i_1 \leq i_2 \leq \cdots \leq i_z \leq \Delta/8$, and $z \geq \Delta/32$. But then, for all $1 \leq a < z$, $m_{i_{a+1}} > m_{i_a} \cdot n^{64/\Delta}$. Therefore, $m_{i_z} > n^{64z/\Delta} \geq n^2$, a contradiction.

Assume now that there are at least $\Delta/32$ type-2 ineligible layers L_i with $1 < i \leq \Delta/8$, and denote their indices by $j_1, j_2, \ldots, j_z$, where $1 < j_1 \leq j_2 \leq \cdots \leq j_z \leq \Delta/8$, and $z \geq \Delta/32$. But then, for all $1 \leq a < z$, $k_{j_{a+1}} > k_{j_a} \cdot k^{64/\Delta}$. Therefore, $k_{j_z} > k^{64z/\Delta} > k$, a contradiction. □

We are now ready to describe the algorithm for ProcCut. The algorithm performs a BFS from the input terminal t in graph G, until it encounters the first index $i > 1$, such that layer L_i is eligible. The algorithm then returns cluster C, which is a subgraph of G induced by $L_1 \cup L_2 \cup \cdots \cup L_{i-1}$, and the set $\hat{T}_C$ of terminals, containing all terminals in $L_1 \cup \cdots \cup L_i$.

We now show that properties (C1) – (C5) hold for this output. Properties (C1), (C3) and (C4) follow immediately from the definition of cluster C and set $\hat{T}_C$ of terminals, and from the fact that, from Claim 3.8, $i \leq \Delta/8$. Property (C2) follows immediately from Condition (L2) in the definition of an eligible layer, since $|T_C| = k_{i-1}$ and $|\hat{T}_C| = k_i$. Lastly, property (C5) follows immediately from the fact that the vertices of C and the terminals of $T \setminus \hat{T}_C$ are separated by layer L_i, so every path connecting a vertex of C and a terminal of $T \setminus \hat{T}_C$ must contain at least $4d$ edges.

Notice that the running time of the algorithm is $O(m_i)$. Since $|E(C)| = m_{i-1}$, from Condition (L1) of an eligible layer, we get that the running time of the algorithm is bounded by $O(m_{i-1} \cdot n^{64/\Delta}) = O(|E(C)| \cdot n^{64/\Delta})$. □

Next, we describe a procedure called ProcPartition, that exploits Procedure ProcCut in order to compute a number of clusters in the input graph G, that contain a large fraction of terminals, such that the diameter of every cluster is relatively small, but pairs of vertices lying in different clusters are sufficiently far away from each other.

3.5.2 Procedure ProcPartition

The input to Procedure ProcPartition consists of an n-vertex graph G, a set $T \subseteq V(G)$ of k vertices called terminals, and distance parameters d and Δ.

The output of the procedure is a collection $\mathcal{C}$ of disjoint clusters of G, and, for every cluster $C \in \mathcal{C}$, a center terminal $t_C \in T_C$, and two sets T'_C, $\hat{T}_C$ of terminals, such that the following properties hold.

R1. for every cluster $C \in \mathcal{C}$, $t_C \in T'_C$; $T'_C \subseteq V(C)$, and $T'_C \subseteq \hat{T}_C$;

R2. for every cluster $C \in \mathcal{C}$, $|\hat{T}_C| \leq |T'_C| \cdot k^{64/\Delta}$;

R3. for every cluster $C \in \mathcal{C}$, $V(C) \subseteq B_G(t_C, \Delta \cdot d)$;

R4. for every cluster $C \in \mathcal{C}$, $\hat{T}_C \subseteq B_G(t_C, \Delta \cdot d)$;

R5. for every pair $C, C' \in \mathcal{C}$ of distinct clusters, for every pair $t' \in T'_C, t'' \in T'_{C'}$ of terminals, $\mathsf{dist}_G(t', t'') \geq d$; and

R6. $\bigcup_{C \in \mathcal{C}} \hat{T}_C = T$.

The following lemma summarizes Procedure ProcCut.

Lemma 3.9 *There is a deterministic algorithm, called Procedure* ProcPartition*, whose input is an n-vertex graph G, a set $T \subseteq V(G)$ of k vertices called terminals, and distance parameters d and Δ. The output of the procedure is a collection $\mathcal{C}$ of disjoint clusters of G, and, for every cluster $C \in \mathcal{C}$, a center terminal $t_C \in T_C$ and sets $T'_C, \hat{T}_C$ of terminals, for which Properties(R1)–(R6) hold. The running time of the procedure is $O(|E(G)| \cdot n^{64/\Delta})$.*

Proof: Throughout the algorithm, we maintain a set $\mathcal{C}$ of disjoint clusters of G, and, for every cluster $C \in \mathcal{C}$, we maintain a center terminal $t_C \in T_C$, and sets $T'_C, \hat{T}_C$ of terminals. We ensure that, throughout the algorithm, properties (R1)–(R5) hold. The algorithm terminates once we achieve Property (R6).

At the beginning of the algorithm, we set $\mathcal{C} = \emptyset$ and $G_0 = G$. We then iterate. In iteration j, we add a new cluster C_j to set $\mathcal{C}$, and define the corresponding terminal t_{C_j} and sets T'_{C_j}, $\hat{T}_{C_j}$ of terminals. We denote $G_j = G \setminus (V(C_1) \cup V(C_2) \cup \cdots \cup V(C_j))$. As the algorithm progresses, we will also delete some terminals from the set T. Specifically, we set $T^{(0)} = T$, and, for all $j \geq 1$, we set $T^{(j)} = T \setminus \left(\hat{T}_{C_1} \cup \hat{T}_{C_2} \cup \cdots \cup \hat{T}_{C_j}\right)$. We will ensure that the following additional invariants hold at the end of iteration j:

I1. $T^{(j)} \subseteq V(G_j)$, and for every pair $t, t' \in T^{(j)}$ of terminals, if $\mathsf{dist}_{G_j}(t, t') \geq 4d$, then $\mathsf{dist}_G(t, t') \geq 4d$.

I2. for every pair t, t' of terminals with $t \in \bigcup_{j'=1}^{j} T'_{C_{j'}}$ and $t' \in T^{(j)}$, $\mathsf{dist}_G(t, t') \geq d$.

At the beginning of the algorithm, $\mathcal{C} = \emptyset$, $G_0 = G$, and $T_0 = T$. Clearly, Properties (R1)–(R5) and invariants (I1) and (I2) are satisfied. We perform iterations until Property (R6) holds. We now describe the execution of the jth iteration.

Execution of the jth iteration. We assume that we are given a set $\mathcal{C} = \{C_1, \ldots, C_{j-1}\}$ of disjoint clusters, and, for every cluster $C_{j'} \in \mathcal{C}$, a terminal $t_{C_{j'}}$, and sets $T'_{C_{j'}}, \hat{T}_{C_{j'}}$ of terminals, such that Properties (R1)–(R5), and Invariants (I1) and (I2) hold. Recall that $G_{j-1} = G \setminus (V(C_1) \cup V(C_2) \cup \cdots \cup V(C_{j-1}))$ and $T^{(j-1)} = T \setminus \left(\hat{T}_{C_1} \cup \hat{T}_{C_2} \cup \cdots \cup \hat{T}_{C_{j-1}}\right)$. We assume that Property (R6) does not hold, so there must be at least one terminal in $T^{(j-1)}$; we let $t \in T^{(j-1)}$ be any such terminal.

In order to execute the jth iteration, we apply Procedure ProcCut from Lemma 3.7 to graph G_{j-1}, set $T^{(j-1)}$ of terminals, and terminal t, keeping the parameters d and Δ unchanged. We denote by C_j the cluster of G_j that the algorithm returns, by $\hat{T}_{C_j} \subseteq T^{(j-1)}$ the resulting set of terminals, and by $T'_{C_j} = T^{(j-1)} \cap V(C)$. Notice that, equivalently, $T'_{C_j} = T_{C_j} \setminus \left(\hat{T}_{C_1} \cup \hat{T}_{C_2} \cup \cdots \cup \hat{T}_{C_{j-1}}\right)$. We also denote $t_{C_j} = t$, and we add C_j to $\mathcal{C}$, completing the iteration.

Analysis of the jth iteration. We now verify that all required properties hold at the end of iteration j, assuming that they held at the beginning of the iteration. Consider first Properties (R1)–(R4). Let $C \in \mathcal{C}$ be any cluster. If $C \neq C_j$, then these properties clearly continue to hold for C. Assume now that $C = C_j$. Properties (R1) and (R2) immediately follow from Properties (C1) and (C2) that are guaranteed by Procedure ProcCut, and from the definition of the set T'_C of terminals. Properties (R3) and (R4) similarly follow from Properties (C3) and (C4), since, for every pair $x, y \in V(G_j)$, $\mathsf{dist}_{G_{j-1}}(x,y) \geq \mathsf{dist}_G(x,y)$, and so $B_{G_{j-1}}(t_C, \Delta \cdot d) \subseteq B_G(t_C, \Delta \cdot d)$.

Consider now any pair $C_{j'}, C_{j''} \in \mathcal{C}$ of clusters, and a pair $t' \in T'_{C_{j'}}$, $t'' \in T'_{C_{j''}}$ of terminals. If both $j', j'' < j$, then we are guaranteed that $\mathsf{dist}_G(t', t'') \geq d$ from the fact that Property (R5) held at the beginning of the iteration. Assume now w.l.o.g. that $j' < j$ and $j'' = j$. In this case, $t'' \in T^{(j-1)}$, and, since we have assumed that Invariant (I2) held at the beginning of the iteration, we get that $\mathsf{dist}_G(t', t'') \geq d$, establishing Property (R5).

Next, we establish Invariant (I1). Recall that $T^{(j)} = T \setminus \left(\hat{T}_1 \cup \cdots \cup \hat{T}_j\right) = T^{(j-1)} \setminus \hat{T}_j$. Consider some terminal $t' \in T^{(j)}$, and assume for contradicting that $t' \notin V(G_j)$. Recall that $V(G_j) = V(G) \setminus (V(C_1) \cup \cdots \cup V(C_j)) = V(G_{j-1}) \setminus V(C_j)$. From Invariant (I1), since $t' \in T^{(j-1)}$, $t' \in V(G_{j-1})$ must hold. Since $t' \notin V(G_j)$, it must be the case that $t' \in V(C_j)$. However, set T'_{C_j} contains every terminal that lies in $V(C_j) \setminus \left(\hat{T}_{C_1} \cup \cdots \cup \hat{T}_{C_{j-1}}\right)$. Since $t' \notin T_{C_1} \cup \cdots \cup \hat{T}_{C_{j-}}$, it must be the case that $t' \in T'_{C_j}$, and so $t' \in \hat{T}_{C_j}$. But then $t' \notin T^{(j)}$, a contradiction.

Consider now some pair $t, t' \in T^{(j)}$ of terminals, and assume that $\mathsf{dist}_{G_j}(t, t') \geq 4d$. Note that, if $\mathsf{dist}_{G_{j-1}}(t, t') \geq 4d$, then, from the fact that Invariant (I1) held at the beginning of the iteration, we get that $\mathsf{dist}_G(t, t') \geq 4d$ must hold. We now show that $\mathsf{dist}_{G_{j-1}}(t, t') \geq 4d$ must hold. Indeed, assume otherwise. Let P be any path of length less than $4d$ in graph G_{j-1} connecting t to t'. Since this path does not lie in graph G_j, at least one vertex $v \in V(P)$ must lie in $V(G_{j-1}) \setminus V(G_j) = V(C_j)$. Therefore, graph G_{j-1} contains a path $P' \subseteq P$ of length less than $4d$ between a vertex $v \in V(C_j)$ and a terminal $t \in T^{(j-1)} \setminus \hat{T}_{C_j}$. This is impossible from Property (C5) of Procedure ProcCut. Therefore, Invariant (I1) continues to hold at the end of iteration j.

Finally, we establish Invariant (I2). Consider a pair t, t' of terminals with $t \in \bigcup_{j'=1}^{j} T'_{C_{j'}}$ and $t' \in T^{(j)}$. If $t \in \bigcup_{j'=1}^{j-1} T'_{C_{j'}}$, then, since Invariant (I2) held at the beginning of the current iteration, we get that $\mathsf{dist}_G(t, t') \geq d$. Therefore we assume that $t \in T'_{C_j}$. In this case, $t \in V(C_j)$, and $t' \in T^{(j-1)} \setminus \hat{T}_{C_j}$ holds, so from Property (C5) of Procedure ProcCut, $\mathsf{dist}_{G_{j-1}}(t, t') \geq 4d$. From the fact that Invariant I1 held at the beginning of the iteration, we then get that $\mathsf{dist}_G(t, t') \geq 4d$.

We conclude that at the end of iteration j, Properties (R1)–(R5), and Invariants (I1) and (I2) continue to hold.

We terminate Procedure ProcPartition once Property (R6) holds. Since, in every iteration, the number of vertices in the current graph G_j decreases, we are guaranteed that the algorithm indeed terminates.

Running time analysis. Recall that, from Lemma 3.7, the running time of Procedure ProcCut is bounded by $O(|E(C)| \cdot n^{64/\Delta})$, where C is the cluster that the procedure returns. Therefore, for all j, the running time of iteration j is at most $O(|E(C_j)| \cdot n^{64/\Delta})$. The total running time of the algorithm is then bounded by $\sum_{C_j \in \mathcal{C}} O(|E(C_j)| \cdot n^{64/\Delta})$. Since the clusters in $\mathcal{C}$ are disjoint, we get that the total running time of Procedure ProcPartition is bounded by $O(|E(G)| \cdot n^{64/\Delta})$. □

3.5.3 Procedure ProcSeparate

Lastly, we provide Procedure ProcSeparate, whose input is similar to that of Procedure ProcPartition. The goal of the procedure is to either produce two large subsets T_1, T_2 of terminals that are sufficiently far from each other in G, or to compute a single terminal $t \in T$ with set $B_G(t, \Delta \cdot d)$ containing many terminals.

Lemma 3.10 *There is a deterministic algorithm, that we call* ProcSeparate*, whose input is an n-vertex graph G, a set $T \subseteq V(G)$ of k vertices called terminals, distance parameters d and Δ, and an additional parameter $0 < \alpha \leq 1$. The algorithm either computes a terminal $t \in T$ with $|B_G(t, \Delta \cdot d) \cap T| > \alpha k$, or it computes two subsets T_1, T_2 of terminals, with $|T_1| = |T_2|$, such that $|T_1| \geq k^{1-64/\Delta} \cdot \min\left\{(1-\alpha), \frac{1}{3}\right\}$, and for every pair $t \in T_1$, $t' \in T_2$ of terminals, $\mathsf{dist}_G(t, t') \geq d$. The running time of the algorithm is $O(|E(G)| \cdot n^{64/\Delta})$.*

Proof: We use the following simple observation.

Observation 3.11 *There is a deterministic algorithm, that, given a collection $\{k_1, k_2, \ldots, k_r\}$ of non-negative integers with $\sum_{j=1}^{r} k_j = k$ and $\max_j \{k_j\} \leq \alpha k$, computes a partition (J_1, J_2) of the set $J = \{1, \ldots, r\}$ of indices, such that $\sum_{j \in J_1} k_j, \sum_{j \in J_2} k_j \geq k \cdot \min\{(1-\alpha), 1/3\}$. The running time of the algorithm is $O(r)$.*

Proof: Assume w.l.o.g. that $k_1 \geq k_2 \geq \cdots \geq k_r$. Assume first that $k_1 \geq k/3$. In this case, we let $J_1 = \{k_1\}$ and $J_2 = J \setminus J - 1$. Clearly, $\sum_{j \in J_2} k_j \geq \sum_{j=1}^{r} k_j - k_1 \geq (1-\alpha)k$, while $\sum_{j \in J_1} k_j \geq k/3$.

Assume now that $k_1 < k/3$. In this case, we start with $J_1 = J_2 = \emptyset$, and process the indices of J one by one. When index j is processed, we add it to J_1 if $\sum_{j' \in J_1} k_{j'} \leq \sum_{j' \in J_1} k_{j'}$ currently holds, and we add it to J_2 otherwise. It is easy to verify that, at the end of this algorithm, $|\sum_{j' \in J_1} k_{j'} - \sum_{j' \in J_2} k_{j'}| \leq \max_{j' \in J}\left\{k_{j'}\right\} \leq k/3$. Therefore, $\sum_{j \in J_1} k_j, \sum_{j \in J_2} k_j \geq k/3$. □

We are now ready to describe the algorithm for the proof of Lemma 3.10. We start by applying Procedure ProcPartition to graph G, the set T of terminals, and parameters d and Δ. Let $\left(\mathcal{C} = \{C_1, \ldots, C_r\}, \left\{t_{C_j}\right\}_{j=1}^{r}, \left\{\hat{T}'_{C_j}\right\}_{j=1}^{r}, \left\{\hat{T}_{C_j}\right\}_{j=1}^{r}\right)$ be the outcome of the procedure. For $1 \leq j \leq r$, denote $k_j = |\hat{T}_j|$. If there is a cluster $C_j \in \mathcal{C}$ with $k_j > \alpha k$, then we return terminal t_{C_j}; from Property (R4), $\hat{T}_{C_j} \subseteq B_G(t_{C_j}, \Delta \cdot d)$, and so $|B_G(t_C, \Delta \cdot d) \cap T| \geq |\hat{T}_{C_j}| = k_j > \alpha k$ must hold.

Assume now that, for all $1 \leq j \leq r$, $k_j \leq \alpha k$. Using the algorithm from Observation 3.11, we compute a partition (J_1, J_2) of the set $J = \{1, \ldots, r\}$ of indices, such that $\sum_{j \in J_1} k_j, \sum_{j \in J_2} k_j \geq k \cdot \min\{(1-\alpha), 1/3\}$.

Denote $\hat{T}^1 = \bigcup_{j \in J_1} \hat{T}_{C_j}$, and $\hat{T}^2 = \bigcup_{j \in J_2} \hat{T}_{C_j}$. Then $|\hat{T}^1|, |\hat{T}^2| \geq k \cdot \min\{(1-\alpha), 1/3\}$. Lastly, we let $T_1 = \bigcup_{j \in J_1} T'_{C_j}$ and $T_2 = \bigcup_{j \in J_2} T'_{C_j}$. From Property (R5) of ProcPartition, for every pair $t \in T_1$, $t' \in T_2$ of terminals, $\mathsf{dist}_G(t, t') \geq d$. From Property (R2), $|T_1| \geq |\hat{T}^1|/k^{64/\Delta}$, and $|T_2| \geq |\hat{T}^2|/k^{64/\Delta}$. We conclude that $|T_1|, |T_2| \geq k^{1-64/\Delta} \cdot \min\left\{(1-\alpha), \frac{1}{3}\right\}$. We discard terminals from the larger of the sets T_1, T_2 as needed, until the cardinalities of both sets become equal. □

3.6 Basic Path Peeling

In this subsection we present an algorithm, that we refer to as ProcPathPeel. This is a simple greedy algorithm for connecting pre-specified pairs of subsets of vertices to each other with short paths.

Similar algorithms were used numerous times before (see e.g. Lemma 6.2 in [CK19], as well as [CS21, Chu21], and Theorems 3.2 and 3.8 in [CGL+20]).

The input to Procedure ProcPathPeel consists of a graph G, collections $A_1, B_1, \ldots, A_k, B_k$ of subsets of its vertices, and parameters $d, \eta > 0$. The output of the procedure is collections $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ of paths in graph G, for which the following properties hold:

- P1. for all $1 \leq i \leq k$, every path in $\mathcal{P}_i$ connects a vertex of A_i to a vertex of B_i, and the endpoints of all paths in $\mathcal{P}_i$ are distinct;
- P2. every path in set $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}_i$ has length at most d, and the paths in $\mathcal{P}$ cause congestion at most η; and
- P3. let E' be the set of all edges $e \in E(G)$, such that exactly η paths of $\mathcal{P}$ use e. For all $1 \leq i \leq k$, let $A'_i \subseteq A_i$, $B'_i \subseteq B_i$ be the sets of vertices that do not serve as endpoints of the paths in $\mathcal{P}_i$. Then for all $1 \leq i \leq k$, $\mathsf{dist}_{G \setminus E'}(A'_i, B'_i) > d$.

We note that we allow a path of P_i to contain vertices of $A_i \cup B_i$, and also vertices from other sets $A_j \cup B_j$ as inner vertices. The following simple lemma summarizes our algorithm for ProcPathPeel.

Lemma 3.12 *There is a deterministic algorithm, called* ProcPathPeel*, that, given a graph G, collections $A_1, B_1, \ldots, A_k, B_k$ of subsets of its vertices, and parameters $d, \eta > 0$, outputs sets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ of simple paths in G, for which properties (P1) – (P3) hold. The running time of the algorithm is $O(m\eta + mdk \log n)$, where $m = |E(G)|$ and $n = V(G)$.*

Proof: We use a simple greedy algorithm combined with the ES-Tree data structure. The algorithm consists of k phases, where for all $1 \leq i \leq k$, we construct the set $\mathcal{P}_i$ of paths in phase i. At the beginning of the algorithm, we set, for all $1 \leq i \leq k$, $\mathcal{P}_i = \emptyset$. Throughout the algorithm, we maintain the set $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}_i$ of paths (that is set to $\emptyset$ at the beginning), and, for every edge $e \in E(G)$, we maintain a counter $n(e)$, whose value is equal to the number of paths in $\mathcal{P}$ containing e. At the beginning of the algorithm, we initialize $n(e) = 0$ for every edge e.

We now describe the execution of the ith phase, for some $1 \leq i \leq k$. We start by constructing a graph G_i. Initially, we let $G_i = G$. We then delete from G_i every edge $e \in E(G)$ with $n(e) = \eta$. Additionally, we add a source vertex s_i to G_i, that connects with an edge to every vertex of A_i, and a destination vertex t_i, that connects with an edge to every vertex of B_i. We also initialize an ES-Tree data structure $\mathcal{T}_i$ in graph G_i, with source vertex s_i, and distance bound $d + 2$.

We then perform iterations, as long as the distance between s_i and t_i in $\mathcal{T}_i$ is bounded by $d+2$. In every iteration, we use the ES-Tree data structure $\mathcal{T}_i$ to compute the shortest s_i-t_i path P in the current graph G_i, whose length must be at most $d+2$. We delete the first and the last edges of P, obtaining a path P' of length at most d, that connects some vertex $x \in A_i$ to some vertex $y \in B_i$. We add path P' to $\mathcal{P}_i$, and increase the counter $n(e)$ for every edge $e \in E(P')$. We then update the current graph G_i, by deleting the edges $(s_i, x), (y, t_i)$ from it, as well as every edge e whose counter $n(e)$ has reached η. The ES-Tree data structure $\mathcal{T}_i$ is also updated with these deletions. Once the ES-Tree data structure $\mathcal{T}_i$ reports that the distance between s_i and t_i in the current graph G_i is greater than $d + 2$, the phase terminates. This completes the description of a phase, and of the algorithm.

From the description of the algorithm, it is immediate to verify that properties (P1) and (P2) hold for the resulting sets $\mathcal{P}_1, \ldots, \mathcal{P}_k$ of paths. Property (P3) is also easy to establish. Indeed, assume for contradiction that, for some index $1 \leq i \leq k$, there is a path P' of length at most d, connecting a

vertex of A'_i to a vertex of B'_i in graph $G \setminus E'$. Then this path must have existed in graph G_i at the end of the ith phase, and so the phase should not have terminated when it did.

Lastly, we bound the running time of the algorithm. Let $m = |E(G)|$. The time that is required to maintain a single ES-Tree $\mathcal{T}_i$ is bounded by $O(md \log n)$. Additionally, whenever a path P is added to set $\mathcal{P}$, the algorithm spends $O(|E(P)|)$ time on processing this path, and on increasing the counters $n(e)$ of edges $e \in E(G)$. Since the counter of an edge may be increased at most η times, the total running time of the algorithm is bounded by $O(m\eta + mkd \log n)$. □

4 The Distanced Matching Game

All graphs discussed in this section are unweighted, so a distance between a pair x, y of vertices is simply the smallest number of edges on any x-y path.

The goal of the Distanced Matching Game is to construct a well-connected graph, that we define next.

Definition 4.1 (Well-connected graph) *Let $G = (V, E)$ be a graph, let $S(G) \subseteq V$ be a subset of its vertices called* supported vertices*, and let $\eta, D > 0$ be parameters. We say that graph G is (η, D)-*well-connected with respect to the set $S(G)$ of supported vertices *if, for every pair $A, B \subseteq S(G)$ of disjoint equal-cardinality subsets of supported vertices, there is a collection $\mathcal{P}(A, B)$ of paths in graph G, routing every vertex of A to a distinct vertex of B (that is, $\mathcal{P}(A, B)$ is a one-to-one routing of A to B), such that the paths in $\mathcal{P}(A, B)$ cause congestion at most η, and the length of every path is at most D.*

Unlike the Cut-Matching Game, whose goal is to construct an expander graph, the goal of the Distanced Matching Game is to construct a graph that is (η, D)-well-connected with respect to a set $S(G)$ of supported vertices; typically, for an n-vertex graph G, we will require that $|S(G)| \geq n - n^{1-\Theta(\epsilon)}$, $\eta \leq n^{O(\epsilon)}$, and $D = 2^{O(1/\epsilon)}$, for a given parameter $0 < \epsilon < 1$.

The main component of the Distanced Matching Game is a *distancing*, that is defined next. Distancings play a role similar to that of cuts in the Cut-Matching Game.

Definition 4.2 (Distancing) *Let G be an n-vertex graph, and let $0 < \delta < 1, d > 0$ be parameters. A (δ, d)-distancing for G is a triple (A, B, E'), where A, B are disjoint subsets of $V(G)$ of cardinality at least $n^{1-\delta}$ each, with $|A| = |B|$, and $E' \subseteq E(G)$ is a subset of edges of cardinality at most $|A|/16$. We require that $\mathsf{dist}_{G \setminus E'}(A, B) \geq d$.*

While the notion of distancing, to the best of our knowledge, was never formally defined before, it is a well-known and widely used fact that one can efficiently obtain a sparse cut in a graph from a distancing. The following lemma, whose analogues have been widely used before, summarizes such an algorithm. The proof uses standard ball-growing technique, and appears in Section A of Appendix.

Lemma 4.1 *There is a deterministic algorithm, whose input consists of a connected graph G with $|V(G)| = n$ and $|E(G)| = m$, a parameter $0 < \varphi < 1/2$, and a (δ, d)-distancing (X, Y, E') in G, where $0 < \delta < 1$ is any parameter, $d \geq (32 \log m)/\varphi$, and $|E'| \leq \varphi |X|/4$. The algorithm computes a cut (X', Y') in graph G, with $X \subseteq X'$ and $Y \subseteq Y'$, such that $|E_G(X', Y')| \leq \varphi \cdot \min\{|E_G(X')|, |E_G(Y')|\}$. The running time of the algorithm is bounded by $O(n + \min\{|E(X')|, |E(Y')|\})$.*

We note that, if the maximum vertex degree in G is bounded by Δ, then $|E(X')| \leq \Delta \cdot |X'|$, and similarly $|E(Y')| \leq \Delta \cdot |Y'|$. Therefore, the algorithm guarantees that $|E(X', Y')| \leq \varphi \cdot \Delta \cdot \min\{|X'|, |Y'|\}$.

A Distanced Matching Game receives as input an integral parameter n, and two other parameters $0 < \delta < 1$ and $d \geq 2^{4/\delta}$. The game is played between a *distancing player* and a *matching player*, in iterations. Over the course of the game, a graph G is constructed. Initially, graph G contains a set V of n vertices and no edges. In every iteration, some edges are added to G.

The ith iteration is executed as follows. First, the distancing player either computes a (δ, d)-distancing (A_i, B_i, E'_i) in the current graph G, or returns "END". If the distancing player returned "END", then the game terminates, and we say that the game lasted for $(i-1)$ iterations. Otherwise, the matching player computes a (possibly partial) matching M_i between vertices of A_i and vertices of B_i, of cardinality at least $|A_i|/8$. We require that M_i does not contain pairs (u, v) of vertices for which edge $(u, v) \in E'$. We note that M_i is not a subset of edges of G; it is just a collection of pairs of vertices from $A_i \times B_i$, with every vertex of $A_i \cup B_i$ appearing in at most one pair in M_i. We add the edges of M_i to graph G, completing iteration i, and proceed to the next iteration.

We note that, once the current graph G contains no (δ, d)-distancing, the game must terminate. From the above description, graph G remains a simple graph (that is, we never add parallel edges to it).

The main technical result of this section is the proof of Theorem 2.1, that bounds the number of iterations in the Distanced Matching Game. We restate the theorem here for convenience.

Theorem 4.2 (Restatement of Theorem 2.1) *Consider a* Distanced Matching Game *with parameters $n > 0, 0 < \delta < 1/4$ and d, such that $d \geq 2^{4/\delta}$ and $n^{\delta} \geq \frac{2^{14} \log n}{\delta^2}$. Then the number of iterations in the game is at most $n^{8\delta}$.*

We note that we do not currently know whether the bounds in this theorem are tight, and in particular whether the requirement that $d \geq 2^{4/\delta}$ is necessary. It would be interesting to establish whether a similar theorem can be proved for values of d that have a lower dependence on $1/\delta$.

We now turn to prove Theorem 4.2. We assume that the paramters n, δ and d are fixed. We let V be a set of n vertices, over which the game is played.

Consider a Distanced Matching Game that lasts for z iterations. We can summarize the game via a *transcript* $\mathbb{T} = ((A_1, B_1, E'_1), M_1, \ldots, (A_z, B_z, E'_z), M_z)$, where for $1 \leq i \leq z$, (A_i, B_i, E'_i) is the distancing computed by the distancing player, and M_i is the matching returned by the matching player in iteration i. For all $1 \leq i \leq z$, we denote by G_i the graph obtained after i iterations of the game, so $V(G_i) = V$ and $E(G_i) = \bigcup_{i'=1}^{i} M_{i'}$. We also let G_0 be the graph with $V(G_0) = V$ and $E(G_0) = \emptyset$.

Consider now any subset $I \subseteq \{1, \ldots, z\}$ of indices, and assume that $I = \{i_1, i_2, \ldots, i_q\}$ with $i_1 < i_2 < \cdots < i_q$. Consider now the following sequence, that, intuitively corresponds to only executing the iterations of the Distanced Matching Game whose indices lie in I:

$$\mathbb{T}' = \left((A_{i_1}, B_{i_1}, E''_{i_1}), M_{i_1}, \ldots, (A_{i_q}, B_{i_q}, E''_{i_q}), M_{i_q}\right).$$

Here for all $1 \leq j \leq q$, E''_{i_j} is defined to be $E'_{i_j} \cap \left(\bigcup_{j'=1}^{j-1} M_{i_{j'}}\right)$.

We claim that $\mathbb{T}'$ is a valid transcript of a Distanced Matching Game. In order to show this, for all $1 \leq j \leq q$, let G'_j be the graph obtained after j iterations of the game, that is, $V(G'_j) = V$, and $E(G'_j) = \bigcup_{j'=1}^{j} M_{i_{j'}}$. We also let G'_0 be the graph containing the set V of vertices and no edges. It is enough to show that, for all $1 \leq j \leq q$, $(A_{i_j}, B_{i_j}, E''_{i_j})$ is a valid (δ, d)-distancing in graph G'_{j-1}. We now prove that this is indeed the case.

Since $(A_{i_j}, B_{i_j}, E'_{i_j})$ is a valid (δ, d)-distancing in the graph $G_{i_j - 1}$ obtained after $(i_j - 1)$ iterations of

the original Distanced Matching Game, it is enough to show that there is no path of length less than d connecting a vertex of A_{i_j} to a vertex of B_{i_j} in graph $G'_{i_{j-1}} \setminus E''_{i_j}$. Assume for contradiction that such a path P exists. Recall that $G'_{i_{j-1}} \subseteq G_{i_j-1}$, and that every edge $e \in E'_{i_j} \cap E(G'_{i_{j-1}})$ lies in E''_{i_j}. Therefore, path P also lies in graph $G_{i_j-1} \setminus E'_{i_j}$, contradicting the fact that $(A_{i_j}, B_{i_j}, E'_{i_j})$ is a valid (δ, d)-distancing in graph G_{i_j-1}. We conclude that $(A_{i_j}, B_{i_j}, E''_{i_j})$ is a valid (δ, d)-distancing in graph G'_{j-1}, and $\mathbb{T}'$ is a valid transcript of a Distanced Matching Game.

To summarize, we can select a subset of iterations from the transcript of the Distanced Matching Game, and obtain a valid transcript of a Distanced Matching Game, induced by these iterations. We say that the Distanced Matching Game associated with transcript $\mathbb{T}'$ is *defined by the set I of indices*.

For the sake of the proof of Theorem 4.2, it would be convenient for us to assume that the cardinalities of the matchings M_i returned in every iteration of the Distanced Matching Game are roughly the same. In order to do so, we partition the set $I^* = \{1, \ldots, z\}$ of indices into at most $r = \lceil 16\delta \log n \rceil$ subsets, as follows. Recall that for all $1 \leq i \leq z$, we are guaranteed that $|M_i| \geq |A_i|/8 \geq n^{1-\delta}/8$, and clearly $|M_i| \leq n$ must hold. For all $1 \leq j \leq r$, we let $I_j \subseteq I^*$ contain all indices i, for which $\frac{n}{2^j} < |M_i| \leq \frac{n}{2^{j-1}}$. Clearly, there must be an index j, such that $|I_j| \geq \frac{z}{r} = \frac{z}{\lceil 16\delta \log n \rceil}$. From now on we will focus on the Distanced Matching Game that is defined by the set I_j of indices, and we will bound the number of iterations in this game, that we denote by $z' \geq \frac{z}{\lceil 16\delta \log n \rceil}$. For simplicity of notation, for all $1 \leq i \leq z'$, we denote the distancing associated with the ith iteration of the game by (A_i, B_i, E'_i) and the matching associated with iteration i by M_i. As before, we denote by G_0 the graph whose vertex set is V and edge set is empty, and for $1 \leq i \leq z'$, we let G_i be the graph obtained after i iterations of the game, that is, $V(G_i) = V$ and $E(G_i) = M_1 \cup \cdots \cup M_i$. We also denote $I = \{1, \ldots, z'\}$ and $G = G_{z'}$.

Let $E^* = \bigcup_{i=1}^{z'} E'_i$, and for all $1 \leq i \leq z'$, let $M'_i = M_i \setminus E^*$. We also denote $H_0 = G_0$, and for all $1 \leq i \leq z'$, we let H_i be a graph whose vertex set is V, and edge set is $E(G_i) \setminus E^* = \bigcup_{i'=1}^{i} M'_{i'}$. We also denote $H = H_{z'}$. The following observation is immediate from the definitions.

Observation 4.3 *For all $1 \leq i \leq z'$, $(A_i, B_i, \emptyset)$ is a (δ, d)-distancing in graph H_i; in other words, there is no path of length less than d connecting A_i to B_i in H_i.*

Note that:

$$|E(H)| = |E(G) \setminus E^*| = \sum_{i=1}^{z'} |M_i| - \sum_{i=1}^{z'} |E'_i| \geq \sum_{i=1}^{z'} (|M_i| - |E'_i|) \geq \sum_{i=1}^{z'} \frac{|M_i|}{2} \geq \frac{|E(G)|}{2}. \tag{1}$$

We have used the fact that, from the definition of the Distanced Matching Game, for all i, $|M_i| \geq |A_i|/8$, while $|E'_i| \leq |A_i|/16$, so $|E'_i| \leq |M_i|/2$ must hold.

We say that an iteration $i \in I$ is *bad* if $|M'_i| < |M_i|/16$; otherwise, iteration i is good. We let $I^b \subseteq I$ be the set of all indices i, such that the ith iteration is bad, and we let $I^g = I \setminus I$ be the set of all indices of good iterations. We use the following simple observation.

Observation 4.4 $|I^b| \leq 7z'/8$.

Proof: Denote $\frac{|I^b|}{z'} = \beta$, and assume for contradiction that $\beta > 7/8$. Recall that, for all $i \in I$, $\frac{n}{2^j} < |M_i| \leq \frac{n}{2^{j-1}}$. Therefore:

$$\begin{aligned}
|E(H)| &= \sum_{i\in I^b} |M_i'| + \sum_{i\in I^g} |M_i'| \\
&\leq \beta\cdot z'\sum_{i\in I^b}\frac{|M_i|}{16} + (1-\beta)\cdot z'\cdot\sum_{i\in I^g}|M_i| \\
&\leq \beta\cdot z'\cdot\frac{n}{16\cdot 2^{j-1}} + (1-\beta)\cdot z'\cdot\frac{n}{2^{j-1}} \\
&= \frac{2z'n}{2^j} - \frac{\beta z'n}{2^j}\cdot\frac{15}{8} \\
&\leq \frac{2z'n}{2^j} - \frac{z'n}{2^j}\cdot\frac{7}{8}\cdot\frac{15}{8} \\
&\leq \frac{23}{64}\cdot\frac{z'n}{2^j} \\
&< \frac{z'n}{2^{j+1}}.
\end{aligned}$$

On the other hand, from Equation 1:

$$|E(H)| \geq \sum_{i=1}^{z'}\frac{|M_i|}{2} \geq \frac{z'n}{2^{j+1}},$$

a contradiction. We conclude that $\beta \leq 7z'/8$ holds. □

To summarize, so far we have shown that:

$$|I^g| \geq \frac{z'}{8} \geq \frac{z}{256\delta\log n}. \tag{2}$$

In order to bound z, it is now enough to bound the number of good iterations in the game. In order to do so, we partition the game into *phases*, each of which (except for, possibly, the last one), contains exactly $\lceil n^{4\delta}\rceil$ good iterations. It is now enough to prove the following claim.

Lemma 4.5 *The number of phases is bounded by* $\frac{2n^{2\delta}}{\delta}$.

Indeed, assume that Lemma 4.5 holds. Then $|I_g| \leq \frac{2n^{2\delta}}{\delta}\cdot\lceil n^{4\delta}\rceil \leq \frac{4n^{6\delta}}{\delta}$, and, from Equation 2, we get that:

$$z \leq |I^g|\cdot 256\delta\log n \leq 1024n^{6\delta}\log n \leq n^{7\delta},$$

since $n^{\delta} \geq 1024\log n$.

In order to complete the proof of Theorem 4.2, it is now enough to prove Lemma 4.5, which we do next. We denote the number of phases in the game by $\hat{z}$.

For every integer $1 \leq k \leq \hat{z}$, we denote by $\hat{H}^{(k)}$ the graph H at the beginning of the kth phase. In other words, if the last iteration of phase $(k-1)$ is i, then $\hat{H}^{(k)} = H_i$. We will define, for every phase k, a collection $\mathcal{C}^{(k)}$ of disjoint subgraphs of $H^{(k)}$, that we refer to as *clusters*; we also refer to $\mathcal{C}^{(k)}$ as a *clustering* of graph $H^{(k)}$. For all integers $1 \leq s \leq \lceil 1/\delta\rceil$, we let $\mathcal{C}_s^{(k)} \subseteq \mathcal{C}^{(k)}$ be the set of all clusters $C \in \mathcal{C}^{(k)}$, with $n^{(s-1)\delta} < |V(C)| \leq n^{s\delta}$, and let $\mathcal{C}_0^{(k)} \subseteq \mathcal{C}^{(k)}$ be the set of all clusters C containing a

single vertex. For $0 \leq s \leq \lceil 1/\delta \rceil$, we say that a cluster $C \in \mathcal{C}_s^{(k)}$ *lies at level s of the clustering* $\mathcal{C}^{(k)}$. If cluster C lies at level s of $\mathcal{C}^{(k)}$, then we say that every vertex of C lies at level s of $\mathcal{C}^{(k)}$.

We will ensure that the following invariants hold for every integer $1 \leq k \leq \hat{z}$:

I1. $V = \bigcup_{C \in \mathcal{C}^{(k)}} V(C)$;

I2. for all $0 \leq s \leq \lceil 1/\delta \rceil$, if $C \in \mathcal{C}_s^{(k)}$ is a level-s cluster of $\mathcal{C}^{(k)}$, then for every pair $x, y \in V(C)$ of vertices, $\mathsf{dist}_C(x,y) \leq 4^s$ (and in particular $\mathsf{dist}_C(x,y) < d$ must hold, as $d \geq 2^{4/\delta}$); and

I3. if C is a cluster of $\mathcal{C}^{(k)}$, and C' is a cluster of $\mathcal{C}^{(k+1)}$, then either $V(C) \cap V(C') = \emptyset$, or $C \subseteq C'$.

Note that Invariant I3 ensures that, if C' is a cluster of $\mathcal{C}^{(k+1)}$, then either C' is a cluster of $\mathcal{C}^{(k)}$, or C' is obtained by taking the union of several clusters of $\mathcal{C}^{(k)}$, and possibly adding some edges to the resulting graph. In particular, the level of any given vertex $v \in V$ may only grow from phase to phase. Consider now some vertex $v \in V$. If vertex v lies at level s of $\mathcal{C}^{(k)}$, and at level $s' > s$ of $\mathcal{C}^{(k+1)}$, then we say that vertex v was *promoted* by $\mathcal{C}^{(k+1)}$, or that it is promoted during phase k.

Initially, we let $\mathcal{C}^{(1)}$ contain, for every vertex $v \in V$, a separate cluster $C(v)$, consisting of only vertex v iteslf, so $\mathcal{C}^{(1)} = \{C(v) \mid v \in V\}$. Therefore, every vertex lies at level 0 of $\mathcal{C}^{(1)}$. Clearly, as the algorithm progresses, the level of a vertex may be at most $\lceil 1/\delta \rceil$. The key in bounding the number of phases is to show that the clusterings $\mathcal{C}^{(k)}$ can be constructed so that a large number of vertices are promoted in every phase. Since every vertex may only be promoted at most $\lceil 1/\delta \rceil$ times, this will be sufficient in order to bound the number of phases. In order to complete the proof of Lemma 4.5, it is enough to prove the following claim.

Claim 4.6 *Consider some integer $k \geq 1$, such that phase k is not the last phase, and assume that we are given a clustering $\mathcal{C}^{(k)}$ of graph $\hat{H}^{(k)}$, for which Invariants I1 and I2 hold. Then there is a clustering $\mathcal{C}^{(k+1)}$ of graph $\hat{H}^{(k+1)}$, for which Invariants I1 and I2 hold. Additionally, for every pair $C \in \mathcal{C}^{(k)}$, $C' \in \mathcal{C}^{(k+1)}$ of clusters, either $V(C) \cap V(C') = \emptyset$ or $C \subseteq C'$ hold. Lastly, the number of vertices that are promoted in $\mathcal{C}^{(k+1)}$ is at least $n^{1-2\delta}$.*

Note that Lemma 4.5 follows immediately from Claim 4.6. Since every vertex may be promoted at most $\lceil 1/\delta \rceil$ times, and every phase promotes at least $n^{1-2\delta}$ vertices, the total number of phases must be bounded by $\frac{n \cdot \lceil 1/\delta \rceil}{n^{1-2\delta}} \leq \frac{2n^{2\delta}}{\delta}$. We now prove Claim 4.6.

Proof of Claim 4.6. Let $I_k \subseteq I$ be the collection of indices $i \in I$, such that iteration i belongs to phase k, and let $I'_k = I_k \cap I^g$ be the set of indices corresponding to good iterations of phase k. Recall that $|I'_k| = \lceil n^{4\delta} \rceil$. We will use the following simple observation.

Observation 4.7 *Consider an iteration $i \in I_k$, and the corresponding distancing (A_i, B_i, E'_i). Then for every cluster $C \in \mathcal{C}^{(k)}$, either $A_i \cap V(C) = \emptyset$, or $B_i \cap V(C) = \emptyset$. Moreover, if there is a pair $C, C' \in \mathcal{C}^{(k)}$ of clusters, such that some edge $e \in M'_i$ connects a vertex of C to a vertex of C', then for every subsequent iteration $i' > i$, either $A_{i'} \cap (V(C) \cup V(C')) = \emptyset$, or $B_{i'} \cap (V(C) \cup V(C')) = \emptyset$ must hold.*

Proof: We fix an index $i \in I_k$, and consider the corresponding distancing (A_i, B_i, E'_i). From Observation 4.3, $(A_i, B_i, \emptyset)$ is a (δ, d)-distancing in graph H_i. Therefore, if P is any path in H_i connecting a vertex of A_i to a vertex of B_i, then the length of P is at least d. Consider now some cluster $C \in \mathcal{C}^{(k)}$, and assume that $C \in \mathcal{C}_s^{(k)}$, for some $0 \leq s \leq \lceil 1/\delta \rceil$. Assume for contradiction that there is a pair

$u, v \in V(C)$ of vertices, with $u \in A_i$ and $v \in B_i$. From Invariant I2, $\mathsf{dist}_C(u,v) \leq 4^s \leq 4^{\lceil 1/\delta \rceil} < d$ (since $d \geq 2^{4/\delta}$). Since $C \subseteq \hat{H}^{(k)} \subseteq H_i$, this is a contradiction.

Assume now that there is a pair $C, C' \in \mathcal{C}^{(k)}$ of clusters, such that some edge $e \in M'_i$ connects a vertex of C to a vertex of C'. Using the same reasoning as above, and since edge e is added to graph H_{i+1}, we get that, for every pair $u \in V(C), v \in V(C')$ of vertices, $\mathsf{dist}_{H_{i+1}}(u,v) \leq 1 + 2 \cdot 4^{\lceil 1/\delta \rceil} < d$. Therefore, for all $i' > i$, $\mathsf{dist}_{H_{i'}}(u,v) < d$ holds as well. Since, from Observation 4.3, $(A_{i'}, B_{i'}, \emptyset)$ is a (δ, d)-distancing in graph $H_{i'}$, we conclude that at most one of the sets $A_{i'}, B_{i'}$ may contain a vertex of $V(C) \cup V(C')$. □

In the remainder of the proof of Claim 4.6, we consider good iterations $i \in I'_k$. For each such iteration i, we will select a large enough subset $M''_i \subseteq M'_i$ of edges, and integers $0 \leq s', s \leq \lceil 1/\delta \rceil$, so that every edge in M''_i connects a vertex of $\bigcup_{C \in \mathcal{C}^{(k)}_s} V(C)$ to a vertex of $\bigcup_{C' \in \mathcal{C}^{(k)}_{s'}} V(C')$.

We then say that iteration i *belongs to class* (s, s'). Since the number of possible classes is small, we can find a class (s, s') to which many iterations of I'_k belong. Assume w.l.o.g. that $s \geq s'$. We will then use the iterations of I'_k from class (s, s') in order to identify a large number of level-s clusters that can be merged together, so that, on the one hand, Invariants I1 and I2 continue to hold, while, on the other hand, a large number of vertices are promoted. The remainder of the proof of Claim 4.6 consists of three steps. In the first step, we define a subset $M''_i \subseteq M'_i$ of edges for each iteration $i \in I'_k$, and classify the iteration into some class (s, s'). Then in the second step we define a new contracted graph J, representing the clusters of $\mathcal{C}^{(k)}_s \cup \mathcal{C}^{(k)}_{s'}$, where (s, s') is the most common class among the iterations of I'_k. Graph J is then used in Step 3 in order to define the new clustering $\mathcal{C}^{(k+1)}$. We now present each of the three steps in turn.

Step 1: Iteration Classification. Consider an iteration $i \in I'_k$. Since iteration i is good, $|M'_i| \geq \frac{|M_i|}{16} \geq \frac{|A_i|}{128} \geq \frac{n^{1-\delta}}{128}$. Consider now an edge $e = (u, v) \in M'_i$, with $u \in A_i, v \in B_i$. From Invariant I1, there must be clusters $C, C' \in \mathcal{C}^{(k)}$ with $u \in V(C)$ and $v \in V(C')$. Moreover, from Observation 4.7, $C \neq C'$ must hold. We say that edge e is of type (s, s'), for a pair $0 \leq s, s' \leq \lceil 1/\delta \rceil$ of indices, if $u \in \mathcal{C}^{(k)}_s$ and $v \in \mathcal{C}^{(k)}_{s'}$; note that it is possible that $s = s'$. Clearly, there is a pair $0 \leq s, s' \leq \lceil 1/\delta \rceil$ of indices, such that the number of edges of type (s, s') in M'_i is at least $\frac{|M'_i|}{\lceil 1/\delta \rceil^2} \geq \frac{n^{1-\delta} \cdot \delta^2}{512}$. From now on we fix this pair of indices, and we denote by $M''_i \subseteq M'_i$ the set of all edges of type (s, s') in M'_i, so $|M''_i| \geq \frac{n^{1-\delta} \cdot \delta^2}{512}$. We say that iteration i is an iteration of type (s, s').

Note that there must be an ordered pair (s, s') of indices, with $0 \leq s, s' \leq \lceil 1/\delta \rceil$, such the number of good iterations in I'_k that are of type (s, s') is at least: $\frac{|I'_k|}{(1+\lceil 1/\delta \rceil)^2} \geq \frac{\delta^2}{6} \cdot \left\lceil n^{4\delta} \right\rceil$. From now on we fix this pair (s, s') of indices, and we denote by $I''_k \subseteq I'_k$ the set of all good iterations i from phase k that are of type (s, s'), so $|I''_k| \geq \frac{\delta^2}{6} \cdot \left\lceil n^{4\delta} \right\rceil$. We assume w.l.o.g. that $s \geq s'$.

Step 2: Contracted Graph. In this step we construct a weighted contracted graph J, as follows. For every cluster $C \in \mathcal{C}^{(k)}_s \cup \mathcal{C}^{(k)}_{s'}$, we add a vertex $v(C)$ to graph J; we refer to vertices of J as *supernodes*, to distinguish them from vertices of V. The set of edges of J is the union of the sets $\{E_i \mid i \in I''_k\}$ of edges that we define below. We refer to the edges of J as *meta-edges*. For every iteration $i \in I''_k$, every meta-edge $\hat{e} \in E_i$ represents some collection $S(\hat{e}) \subseteq M''_i$ of edges.

Consider some iteration $i \in I''_k$. For every pair $C \in \mathcal{C}^{(k)}_s, C' \in \mathcal{C}^{(k)}_{s'}$ of clusters, such that at least one edge of M''_i connects a vertex of C to a vertex of C', we add a meta-edge $\hat{e} = (v(C), v(C'))$ to E_i. We let $S(\hat{e})$ be the set of all edges of M''_i that connect vertices of C to vertices of C', and we set the weight of the meta-edge $\hat{e}$ to be $w(\hat{e}) = |S(\hat{e})|$. We note that, since M''_i is a matching of cardinality at

least $\frac{n^{1-\delta}\cdot\delta^2}{512}$, we get that $\sum_{\hat{e}\in E_i} w(e) \geq \frac{n^{1-\delta}\cdot\delta^2}{512}$. Moreover, for every cluster $C \in \mathcal{C}_s^{(k)} \cup \mathcal{C}_{s'}^{(k)}$, the total weight of all meta-edges of E_i incident to C is at most $|V(C)|$.

Consider a pair $i, i' \in I_k''$ of indices with $i < i'$. Note that, if a meta-edge $(v(C), v(C'))$ belongs to E_i, then, from Observation 4.7, meta-edge $(v(C), v(C'))$ may not lie in $E_{i'}$. We set $E(J) = \bigcup_{i\in I_k''} E_i$. From the above discussion, graph J contains no parallel edges. For every supernode $v(C) \in V(J)$, the total weight of all meta-edges incident to $v(C)$ is bounded by $|V(C)| \cdot |I_k''|$. The total weight of all meta-edges in graph J is at least $\frac{n^{1-\delta}\cdot\delta^2}{512} \cdot |I_k''|$.

Step 3: Constructing the Clustering $\mathcal{C}^{(k+1)}$. In order to construct clustering $\mathcal{C}^{(k+1)}$, we start with $\mathcal{C}^{(k+1)} = \mathcal{C}^{(k)}$, and then iteratively merge some clusters of $\mathcal{C}^{(k+1)}$. In every iteration, we consider the graph J. Assume that there is a cluster $C' \in \mathcal{C}_{s'}^{(k)}$, such that supernode $v(C')$ has at least n^δ neighbor vertices in graph J. We denote the neighbor vertices of $v(C')$ by $v(C_1), \ldots, v(C_q)$. Note that for all $1 \leq a \leq q$, $C_a \in \mathcal{C}_s^{(k)}$ must hold. We delete clusters $C', C_1, \ldots, C_q$ from $\mathcal{C}^{(k+1)}$, and instead add a single cluster C^*, whose vertex set is $V(C') \cup V(C_1) \cup \cdots \cup V(C_q)$, and edge set is the union of $E(C') \cup E(C_1) \cup \cdots \cup E(C_q)$ with the set $\bigcup_{a=1}^{q} S(v(C'), v(C_a))$ of edges. Note that $C^* \subseteq \hat{H}^{(k+1)}$ holds. We then delete vertices $v(C'), v(C_1), \ldots, v(C_q)$ from graph J.

Observe that, for every vertex $x \in V(C^*)$, the level of x in $\mathcal{C}^{(k)}$ was either s or $s' \leq s$, and, since $|V(C^*)| \geq s \cdot n^\delta$, the level of x in $\mathcal{C}^{(k+1)}$ is at least $s+1$. Therefore, every vertex in $V(C^*)$ is promoted in the current phase. We use the following simple observation, that will allow us to establish Invariant I2.

Observation 4.8 *For every pair $x, y \in V(C^*)$ of vertices, $\mathsf{dist}_{C^*}(x, y) \leq 4^{s+1}$.*

Proof: If both x and y belong to a single cluster of $\{C', C_1, \ldots, C_q\}$, then, since Invariant I2 held for $\mathcal{C}^{(k)}$, and since each such cluster is contained in C^*, $\mathsf{dist}_{C^*}(x, y) \leq 4^s$ must hold. Assume now that x and y belong to different clusters. We assume w.l.o.g. that $x \in V(C_1)$ and $y \in V(C_2)$; the other cases are treated similarly.

From the definition of cluster C^*, meta-edges $\hat{e}_1 = (v(C'), v(C_1))$, $\hat{e}_2 = (v(C'), v(C_2))$ lie in J. Consider any real edge $e_1 \in S(\hat{e}_1)$ and $e_2 \in S(\hat{e}_2)$. Both edges must lie in $\hat{H}^{(k+1)}$, and in C^*. We denote $e_1 = (x_1, y_1)$ with $x_1 \in V(C_1)$, and $e_2 = (x_2, y_2)$ with $y_2 \in V(C_2)$. In particular, $y_1, x_2 \in V(C')$ must hold. From Invariant I2, there is a path P_1 of length at most 4^s connecting x to x_1 in C_1; a path P' of length at most $4^{s'} \leq 4^s$ connecting y_1 to x_2 in C'; and a path P_3 of length at most 4^s connecting y_2 to y in C_2. By combining these three paths with edges e_1 and e_2, we obtain a path in cluster C^*, connecting x to y, whose length is at most $3 \cdot 4^s + 1 \leq 4^{s+1}$. Therefore, $\mathsf{dist}_{C^*}(x, y) \leq 4^{s+1}$. □

The algorithm terminates once, every supernode $v(C')$ of J with $C' \in \mathcal{C}_{s'}^{(k)}$, the number of meta-edges incident to $v(C')$ in the current graph J is less than n^δ.

From the above discussion, once the algorithm terminates, Invariant I2 holds for the final clustering $\mathcal{C}^{(k+1)}$, as every newly added cluster to $\mathcal{C}^{(k+1)}$ belongs to level at least $(s+1)$. It is immediate to verify that Invariant I1 holds for the final set $\mathcal{C}^{(k+1)}$ of clusters. It now only remains to prove that sufficiently many vertices are promoted in the current iteration.

Consider the graph J that is obtained at the end of the algorithm. In this graph, for every cluster $C' \in \mathcal{C}_{s'}^{(k)}$, its corresponding supernode $v(C')$ has fewer than n^δ meta-edges incident to $v(C')$. Since, for every meta-edge $\hat{e} \in E(J)$ that is incident to a supernode $v(C)$, $w(\hat{e}) \leq |V(C)|$ must hold, we get that the total weight of all edges remaining in graph J at the end of the algorithm is bounded by:

$$\sum_{C' \in \mathcal{C}^{(k)}_{s'}} |V(C')| \cdot n^{\delta} \leq n^{1+\delta}.$$

Recall that the total weight of all meta-edges of J at the beginning of the algorithm was at least:

$$\frac{n^{1-\delta} \cdot \delta^2}{512} \cdot |I''_k| \geq \frac{n^{1+3\delta} \cdot \delta^4}{2^{12}} > 2n^{1+\delta},$$

since $|I''_k| \geq \frac{\delta^2}{6} \cdot \lceil n^{4\delta} \rceil$ and $n^{\delta} \geq 2^{14}/\delta^4$.

Therefore, the total weight of the meta-edges that remain at the end of the algorithm in J is less than half the original total weight. In other words, the total weight of all meta-edges that were deleted from J is at least $\frac{n^{1-\delta} \cdot \delta^2}{1024} \cdot |I''_k|$. Each of the deleted meta-edges is incident to some supernode $v(C)$, with $C \in \mathcal{C}^{(k)}_s$ that was deleted from J. Recall that every vertex of such a cluster C is promoted in the current phase.

Consider now some supernode $v(C)$ that was deleted from J in the current phase. The total weight of all meta-edges incident to $v(C)$ in the original graph J was at most $|V(C)| \cdot |I''_k|$, and each of the vertices of C was promoted in phase k. Therefore, if we denote by U is the set of all vertices of V that were promoted in phase k, then the total weight of all meta-edges that were deleted from J is at most $|U| \cdot |I''_k|$. Since, as shown above, the total weight of all such edges is at least $\frac{n^{1-\delta} \cdot \delta^2}{1024} \cdot |I''_k|$, and since we have assumed that $n^{\delta} \geq 2^{14}/\delta^2$, we get that $|U| \geq \frac{n^{1-\delta} \cdot \delta^2}{1024} \geq n^{1-2\delta}$. □

This concludes the proof of Theorem 4.2. We obtain the following immediate corollary of the theorem.

Corollary 4.9 *Consider a* Distanced Matching Game *with parameters $n > 0, 0 < \delta < 1/4$ and d, such that $d \geq 2^{4/\delta}$, $n^{\delta} \geq \frac{2^{14} \log n}{\delta^2}$. Let G be the graph that is obtained at the end of the game. Then $|E(G)| \leq n^{1+8\delta}$ holds, and every vertex of G has degree at most $n^{8\delta}$.*

The corollary follows from the fact that the set $E(G)$ of edges is partitioned into at most $n^{8\delta}$ matchings – the responses of the matching player in the game.

5 Hierarchical Support Structure

A Hierarchical Support Structure uses two parameters, an integer $N > 0$, and another parameter $0 < \epsilon \leq 1/4$. Throughout, we denote $\epsilon' = \epsilon^4$. For an integer $j \geq 1$, we also let $\eta_j = N^{6+256j\epsilon^2}$ and $\tilde{d}_j = 2^{cj/\epsilon^4}$, where c is a sufficiently large constant.

For all $1 \leq j \leq \lceil 1/\epsilon \rceil$, we define a *level-$j$* Hierarchical Support Structure for a graph containing N^j vertices. The definition of the support structure is recursive.

Level-1 Hierarchical Support Structure. Given a graph H with $|V(H)| = N$, level-1 Hierarchical Support Structure for H consists of a subset $S(H)$ of vertices of H, such that $|V(H) \setminus S(H)| \leq N^{1-\epsilon^4}$.

Level-j Hierarchical Support Structure. Consider now some integer $1 < j \leq \lceil 1/\epsilon \rceil$. Let H be a graph with $|V(H)| = N^j$. A *level-j Hierarchical Support Structure* for graph H consists of the following:

- a collection $\mathcal{H} = \{H_1, \ldots, H_r\}$ of $r = N - \left\lceil 2N^{1-\epsilon^4} \right\rceil$ graphs, such that all vertices in sets $V(H_1), V(H_2), \ldots, V(H_r)$ are mutually disjoint, and additionally, for all $1 \leq i \leq r$: $V(H_i) \subseteq V(H)$; $|V(H_i)| = N^{j-1}$; and $|E(H_i)| \leq N^{j-1+32\epsilon^2}$ hold;
- an embedding of the graph $H' = \bigcup_{i=1}^{r} H_i$ into graph H via paths of length at most $2^{64/\epsilon^4}$, that causes congestion at most $N^{128\epsilon^2}$;
- for all $1 \leq i \leq r$, a level-$(j-1)$ Hierarchical Support Structure for graph H_i; and
- a set $S(H) \subseteq V(H)$ of vertices, where $S(H) = \bigcup_{H_i \in \mathcal{H}} S(H_i)$, and, for all $H_i \in \mathcal{H}$, $S(H_i)$ is the set of vertices that is given as part of the level-$(j-1)$ Hierarchical Support Structure for H_i.

Additionally, we require every graph $H_i \in \mathcal{H}$ is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected with respect to the set $S(H_i)$ of vertices. We say that $\mathcal{H}$ is the set of graphs associated with the level-j Hierarchical Support Structure for graph H.

This completes the definition of a Hierarchical Support Structure. We will need to use the following simple claim.

Claim 5.1 *Let $1 \leq j \leq \lceil 1/\epsilon \rceil$ be an integer, and let H be a graph with $|V(H)| = N^j$, together with a level-j Hierarchical Support Structure. Then $|V(H) \setminus S(H)| \leq |V(H)| \cdot \frac{4j}{N^{\epsilon^4}}$.*

Proof: The proof is by induction on j. When $j = 1$, then $|V(H)| = N$, and, from the definition of level-1 Hierarchical Support Structure, $|V(H) \setminus S(H)| \leq N^{1-\epsilon^4} = \frac{|V(H)|}{N^{\epsilon^4}} \leq |V(H)| \cdot \frac{4j}{N^{\epsilon^4}}$.

Consider now some integer $j > 1$, and assume that the claim holds for $j-1$. Let H be a graph with $|V(H)| = N^j$, for which a level-j Hierarchical Support Structure is given, and let $\mathcal{H}$ be the collection of graphs associated with the structure. Let $V_1 = \bigcup_{H_i \in \mathcal{H}} V(H_i)$ and $V_2 = V(H) \setminus V_1$. From the definition of level-j Hierarchical Support Structure, $|\mathcal{H}| = N - \left\lceil 2N^{1-\epsilon^4} \right\rceil \geq N - 4N^{1-\epsilon^4}$. Therefore, $|V_1| \geq N^j - \frac{4N^j}{N^{\epsilon^4}}$, and $V_2 \leq \frac{4N^j}{N^{\epsilon^4}} = \frac{4|V(H)|}{N^{\epsilon^4}}$.

Let $V_1' = \bigcup_{H_i \in \mathcal{H}} S(H_i)$ and $V_1'' = V_1 \setminus V_1'$. Clearly, $V(H) \setminus S(H) = V_1'' \cup V_2$. We now bound $|V_1''|$.

Since, for every graph $H_i \in \mathcal{H}$, by the induction hypothesis, $|V(H_i) \setminus S(H_i)| \leq |V(H_i)| \cdot \frac{4(j-1)}{N^{\epsilon^4}} = \frac{4(j-1) \cdot N^{j-1}}{N^{\epsilon^4}}$, and since $|\mathcal{H}| < N$, we get that $|V_1''| \leq \frac{4(j-1) \cdot N^j}{N^{\epsilon^4}} = \frac{4(j-1) \cdot |V(H)|}{N^{\epsilon^4}}$.

Altogether, we get that:

$$|V(H) \setminus S(H)| = |V_1''| + |V_2| \leq \frac{4(j-1) \cdot |V(H)|}{N^{\epsilon^4}} + \frac{4|V(H)|}{N^{\epsilon^4}} = \frac{4j \cdot |V(H)|}{N^{\epsilon^4}}$$

□

The following theorem provides an algorithm for the Distancing Player in the Distanced Matching game.

Theorem 5.2 *There is a large enough constant c, and a deterministic algorithm, whose input consists of a parameter $0 < \epsilon < 1/4$, a pair N, $1 \leq j \leq \lceil 1/\epsilon \rceil$ of integers, and a graph H with $|V(H)| = N^j$, such that N is sufficiently large, so that $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^5}$ holds. The algorithm computes one of the following:*

- *either a (δ_j, d)-distancing (A, B, E') in graph H, where $\delta_j = 4j\epsilon'$, $d = 2^{32/\epsilon^4}$ and $|E'| \leq \frac{|A|}{N^{j\epsilon^4}}$; or*

- *a level-j Hierarchical Support Structure for H, such that graph H is $(\eta_j, \tilde{d}_j)$-well-connected with respect to the set $S(H)$ of vertices defined by the support structure, where $\eta_j = N^{6+256j\epsilon^2}$ and $\tilde{d}_j = 2^{cj/\epsilon^4}$.*

The running time of the algorithm is bounded by:

$$cj \cdot N^{j(1+64\epsilon^2)+7} + c|E(H)| \cdot N^6.$$

We prove Theorem 5.2 in Section 6. Note that Theorem 2.2 follows from the theorem directly, by setting $j = 1/\epsilon$.

The following immediate corollary of the theorem can be used in order to either embed a large graph H into an input graph G, and construct a Hierarchical Support Structure for H, so that graph H is well-connected with respect to the resulting set $S(H)$ of vertices given by the Hierarchical Support Structure; or to compute a distancing in graph G. The latter can in turn be used in order to compute a sparse cut in G, via Lemma 4.1. We state the corollary in a slightly more general form that will be helpful for us later: we assume that, together with graph G, we are given a subset T of its vertices called terminals, and that we are interested in embedding a large graph H into G with $V(H) \subseteq T$.

Corollary 5.3 *There is a deterministic algorithm, whose input consists of an n-vertex graph G, a set T of k vertices of G called terminals, and parameters $\frac{2}{(\log k)^{1/12}} < \epsilon < \frac{1}{400}$, $d > 1$ and $\eta > 1$, such that $1/\epsilon$ is an integer. The algorithm computes one of the following:*

- *either a pair $T_1, T_2 \subseteq T$ of disjoint subsets of terminals, and a set E' of edges of G, such that:*
  - *$|T_1| = |T_2|$ and $|T_1| \geq \frac{k^{1-4\epsilon^3}}{4}$;*
  - *$|E'| \leq \frac{d \cdot |T_1|}{\eta}$; and*
  - *for every pair $t \in T_1, t' \in T_2$ of terminals, $\mathsf{dist}_{G \setminus E'}(t, t') > d$;*
- *or a graph H with $V(H) \subseteq T$, $|V(H)| = N^{1/\epsilon} \geq k - k^{1-\epsilon/2}$, where $N = \lfloor k^\epsilon \rfloor$, and maximum vertex degree at most $k^{32\epsilon^3}$, together with an embedding $\mathcal{P}$ of H into G via paths of length at most d that cause congestion at most $\eta \cdot k^{32\epsilon^3}$, and a level-$(1/\epsilon)$ Hierarchical Support Structure for H, such that H is $(\eta', \tilde{d})$-well-connected with respect to the set $S(H)$ of vertices defined by the support structure, where $\eta' = N^{6+256\epsilon}$, and $\tilde{d} = 2^{c/\epsilon^5}$, with c being the constant used in the definition of the Hierarchical Support Structure.*

The running time of the algorithm is $O\left(k^{1+O(\epsilon)} + |E(G)| \cdot k^{O(\epsilon^3)} \cdot (\eta + d \log n)\right)$.

Proof: Let $q = 1/\epsilon$, let $N = \lfloor k^\epsilon \rfloor$, and let $T' \subseteq T$ be any subset of N^q terminals. Observe that:

$$N^q = \lfloor k^\epsilon \rfloor^{1/\epsilon} \geq (k^\epsilon - 1)^{1/\epsilon} = k \cdot \left(1 - \frac{1}{k^\epsilon}\right)^{1/\epsilon} \geq k \cdot \left(1 - \frac{1}{k^\epsilon \cdot \epsilon}\right) \geq k - \frac{k^{1-\epsilon}}{\epsilon} \geq k - k^{1-\epsilon/2} \qquad (3)$$

(we have used the fact that for all $0 < \delta < 1$ and $a > 2$, $(1-\delta)^a \geq 1 - \delta a$).

We will attempt to construct a graph H with $V(H) = T'$, together with an embedding $\mathcal{P}$ of H into G with congestion at most $\eta \cdot k^{32\epsilon^3}$ and path lengths at most d, and a level-q Hierarchical Support Structure for H, such that H is $(\eta', \tilde{d})$-well-connected with respect to the set $S(H)$ of vertices defined

by the support structure. If we fail to construct such a graph H, we will compute the sets $T_1, T_2 \subseteq T$ of terminals and the set $E' \subseteq E(G)$ of edges as required.

We will employ the distanced-matching game on graph H with parameters $n = |T'|$, $\delta = 4\epsilon^3$, and distance parameter $d' = 2^{32/\epsilon^4}$.

We will use the algorithm from Theorem 5.2 for the distancing player, with parameter $j = q$, and parameters N and ϵ remaining unchanged. In order to be able to use the algorithm, we need to verify that $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^5}$ holds. Recall that, from the conditions of Corollary 5.3, $\frac{2}{(\log k)^{1/12}} < \epsilon < \frac{1}{400}$. Therefore, $\log k > (2/\epsilon)^{12}$ and $k > 2^{(2/\epsilon)^{12}}$. Moreover, from the above calculations, $\frac{\log k}{\log\log k} > \frac{1}{\epsilon^8}$ holds, and so $k^{\epsilon^8} > \log k$ must hold. Altogether, we get that:

$$\frac{N^{\epsilon^4}}{\log N} \geq \frac{k^{\epsilon^5}}{2 \cdot \log k} \geq k^{\epsilon^6} \geq 2^{128/\epsilon^5}. \tag{4}$$

We will bound the number of iterations of the Distanced Matching game via Theorem 4.2. In order to use Theorem 4.2, we need to verify that $\frac{|T'|^\delta}{\log(|T'|)} \geq \frac{2^{14}}{\delta^2}$, or, equivalently: $\frac{|T'|^{4\epsilon^3}}{\log(|T'|)} \geq \frac{2^{10}}{\epsilon^6}$. Recall that $|T'| = N^q = N^{1/\epsilon}$, and so $\frac{|T'|^{4\epsilon^3}}{\log(|T'|)} = \frac{\epsilon \cdot N^{4\epsilon^2}}{\log N} \geq \frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^5} \geq \frac{2^{10}}{\epsilon^6}$ from Equation 4, and since $\epsilon \leq 1/400$. We also need to verify that $d' \geq 2^{4/\delta}$. Since $d' = 2^{32/\epsilon^4}$ and $\delta = 4\epsilon^3$, this is immediate to verify. From Theorem 4.2, we can now conclude that the number of iterations in a Distanced Matching game with parameters $n = |T'|$, $\delta = 4\epsilon^3$, and distance parameter $d' = 2^{32/\epsilon^4}$ is bounded by $|T'|^{8\delta} \leq k^{32\epsilon^3}$.

We start with a graph H whose vertex set is $V(H) = T'$, and edge set is $E(H) = \emptyset$, and then iterate. In each iteration i, we will add some set E_i of edges to graph H, and we will define an embedding $P(e)$ for every edge $e \in E_i$. We now describe the execution of a single iteration.

Execution of Iteration i.

We apply the algorithm from Theorem 5.2 to the current graph H, with parameter $j = q$, and parameters N, ϵ remaining unchanged. Note that $|V(H)| = |T'| = N^q$, and, as we have established above, $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^5}$ holds.

We now consider two cases. The first case is when the algorithm from Theorem 5.2 returns a (δ_q, d')-distancing (X_i, Y_i, E_i') in graph H, where $\delta_q = 4j\epsilon' = 4q\epsilon^4 = 4\epsilon^3 = \delta$ (since $q = 1/\epsilon$), and $d' = 2^{32/\epsilon^4}$. In this case, we say that iteration i is *regular*. We view this distancing as the response of the distancing player in iteration i of the Distanced Matching game that we play on graph H.

We then apply Procedure ProcPathPeel from Lemma 3.12 to graph G and sets $A_1 = X_i, B_1 = Y_i$ of its vertices, together with parameters d and η from the statement of Corollary 5.3. Let $\mathcal{Q}_1$ denote the collection of paths that the algorithm returns. Recall that every path in $\mathcal{Q}_1$ connects some vertex of X_i to a vertex of Y_i, and that every vertex of $X_i \cup Y_i$ may serve as an endpoint of at most one such path. We let $M_i \subseteq X_i \times Y_i$ be the matching that is defined by the paths in $\mathcal{Q}_1$: a pair (x, y) of vertices with $x \in X_i, y \in Y_i$ is added to M_i iff some path $Q(x, y) \in \mathcal{Q}_1$ has endpoints x, y. We again consider two cases. The first case happens if $|M_i| \geq |X_i|/2$. In this case, we say that iteration i is successful. We obtain a collection $E_i \subseteq M_i$ of edges as follows: we start with $E_i = M_i$, and we delete from E_i all pairs (x, y) of vertices where edge (x, y) lies in E_i'. Since, from the definition of distancing, $|E_i'| \leq |X_i|/16$, we get that $|E_i| \geq |X_i|/4$. We let $\mathcal{P}_i = \{Q(x, y) \mid (x, y) \in E_i\}$ be the collection of paths that route the pairs of vertices in E_i. We add the edges of E_i to graph H, and we view E_i as

the response of the matching player in iteration i. We also view the set $\mathcal{P}_i$ of paths in graph H as an embedding of the set E_i of edges. Recall that each path in $\mathcal{P}_i$ has length at most d, and the paths in $\mathcal{P}_i$ cause congestion at most η. We then continue to the next iteration.

The second case happens if $|M_i| < |X_i|/2$. In this case we say that iteration i is unsuccessful. Let E' be the set of all edges e in graph G that participate in exactly η paths in $\mathcal{Q}_1$. Let $X' \subseteq X_i$ and $Y' \subseteq Y'$ be the sets of vertices that do not serve as endpoints of the paths in $\mathcal{Q}_1$. Recall that Lemma 3.12 guarantees (via Property P3) that the length of the shortest path connecting a vertex of X' to a vertex of Y' in $G \setminus E'$ is greater than d.

Recall that $|X'| = |Y'| \geq \frac{|X_i|}{2} \geq \frac{|T'|^{1-\delta}}{2} \geq \frac{k^{1-\delta}}{4} \geq \frac{k^{1-4\epsilon^3}}{4}$. Since every path in $\mathcal{Q}_1$ has length at most d, we get that $\sum_{Q \in \mathcal{Q}_1} |E(Q)| \leq d \cdot \frac{|X_i|}{2} \leq d \cdot |X'|$. Since set E' contains edges that participate in η paths in $\mathcal{Q}_1$, we get that: $|E'| \leq \frac{d \cdot |X'|}{\eta}$. We return the set E' of edges and the sets $T_1 = X'$, $T_2 = Y'$ of terminals. From the above discussion, $|T_1| = |T_2|$, $|T_1| \geq \frac{k^{1-4\epsilon^3}}{4}$, and $|E'| \leq \frac{d \cdot |T_1|}{\eta}$ hold as required. Moreover, for every pair $t \in T_1, t' \in T_2$ of terminals, $\mathsf{dist}_{G \setminus E'}(t, t') > d$.

It remains to consider the second case, when the algorithm from Theorem 5.2 constructs a level-q Hierarchical Support Structure for H, such that graph H is $(\eta_q, \tilde{d}_q)$-well-connected with respect to the set $S(H)$ of vertices defined by the support structure, where $\tilde{d}_q = 2^{cq/\epsilon^4} = 2^{c/\epsilon^5} = \tilde{d}$, and:

$$\eta_q = N^{6+256q\epsilon^2} = N^{6+256\epsilon} = \eta'.$$

In this case, we say that iteration i is irregular. Recall that $|V(H)| = |T'| = N^q \geq k - k^{1-\epsilon/2}$ from Inequality 3. As observed already, the number of iterations in the Distanced Matching game is bounded by $k^{32\epsilon^3}$, and so the maximum vertex degree in H is bounded by $k^{32\epsilon^3}$, and the number of edges in graph H is bounded by $k^{1+32\epsilon^3}$ throughout the algorithm. If we denote by $z \leq k^{32\epsilon^3}$ the number of iterations in the Distanced Matching game, then for all $1 \leq i \leq z$, we have constructed a set $\mathcal{P}_i$ of paths in graph H embedding the edges of E_i. The paths in $\mathcal{P}_i$ have length at most d each, and they cause congestion at most η in G. By letting $\mathcal{P} = \bigcup_{i=1}^{z} \mathcal{P}_i$, we obtain an embedding of graph H into G via paths of length at most d, that cause congestion at most $\eta \cdot z \leq \eta \cdot k^{32\epsilon^3}$. We output graph H, its embedding $\mathcal{P}$, and the level-q Hierarchical Support Structure for H.

This completes the description of a single iteration. It now remains to bound the running time of the algorithm.

Running Time Analysis

Recall that, throughout the algorithm, $|E(H)| \leq k^{1+32\epsilon^3}$ holds, and that the number of regular successful iterations in the algorithm is at most $k^{32\epsilon^3}$. Additionally, there could be at most one irregular iteration, and at most one regular but unsuccessful iteration.

We now bound the running time of a single iteration. This running time is dominated by the running times of the algorithms from Theorem 5.2 and Lemma 3.12.

The former is bounded by:

$$O(q \cdot N^{q(1+64\epsilon^2)+7} + |E(H)| \cdot N^6) \leq O\left(k^{1+O(\epsilon)} + k^{1+32\epsilon^3} \cdot k^{O(\epsilon)}\right) \leq O\left(k^{1+O(\epsilon)}\right).$$

The latter is bounded by: $O(|E(G)|(\eta + d \log n))$.

The total running time of the algorithm is then bounded by:

$$O\left(k^{1+O(\epsilon)} + |E(G)| \cdot k^{O(\epsilon^3)} \cdot (\eta + d \log n)\right).$$

□

6 Algorithm for the Distancing Player – Proof of Theorem 5.2

This section is dedicated to proving Theorem 5.2. Throughout the proof we use the following three parameters: $\epsilon' = \epsilon^4$; $\Delta = 64/\epsilon'$; and $d' = 2d \cdot \Delta$. Note that:

$$d' = 2\Delta d = \frac{128d}{\epsilon'} = \frac{128 \cdot 2^{32/\epsilon^4}}{\epsilon^4} \leq 2^{64/\epsilon^4} < \frac{1}{4} \cdot 2^{c/\epsilon^4}, \tag{5}$$

since c is large enough.

The proof is by induction on j. We start with the base case, where $j = 1$.

Base Case: $j = 1$

We assume that we are given a graph H on N vertices. Our goal is to either compute a (δ_1, d)-distancing in graph H, or to construct a set $S(H) \subseteq V(H)$ of vertices, such that $|V(H) \setminus S(H)| \leq N^{1-\epsilon^4}$, and graph H is $(\eta_1, \tilde{d}_1)$-well-connected with respect to $S(H)$.

We apply Algorithm ProcSeparate from Lemma 3.10 to graph H, with the set $T = V(H)$ of terminal vertices, with distance parameters d and Δ remaining the same, and parameter $\alpha = 1 - \frac{1}{N^{\epsilon'}}$.

Assume first that the outcome of the algorithm is a pair $T_1, T_2 \subseteq V(H)$ of subsets of vertices, with $|T_1| = |T_2|$, such that for every pair $t \in T_1$, $t' \in T_2$ of vertices, $\mathsf{dist}_H(t, t') \geq d$, and additionally:

$$|T_1| \geq N^{1-64/\Delta} \cdot \min\left\{(1-\alpha), \frac{1}{3}\right\} \geq N^{1-2\epsilon'}.$$

(recall that $\Delta = 64/\epsilon'$ and $N^{\epsilon'} > 3$). In this case, since $\delta_1 = 4\epsilon'$, we obtain a (δ_1, d)-distancing $(T_1, T_2, \emptyset)$ in graph H. We return this distancing as the outcome of the algorithm.

Otherwise, Procedure ProcSeparate must return a vertex $v \in V(H)$, with $|B_H(v, \Delta \cdot d)| > \alpha \cdot N = N \cdot \left(1 - \frac{1}{N^{\epsilon'}}\right)$. In this case, we set $S(H) = B_H(v, \Delta \cdot d)$, and we report that graph H is $(\eta_1, \tilde{d}_1)$-well-connected with respect to $S(H)$. We also return $S(H)$ as level-1 Hierarchical Support Structure for H. Note that $|V(H) \setminus S(H)| \leq N^{1-\epsilon'} = N^{1-\epsilon^4}$ as required. It remains to show that graph H is indeed $(\eta_1, \tilde{d}_1)$-well-connected with respect to $S(H)$. Recall that $\eta_1 > N$ and $d' < \tilde{d}_1$ from Inequality 5, since $\tilde{d}_1 = 2^{c/\epsilon^4}$. Let $A, B \subseteq S(H)$ be any pair of equal-cardinality subsets of vertices. We define an arbitrary perfect matching $M \subseteq A \times B$ between vertices of A and vertices of B. Consider now any pair $(a, b) \in M$ of matched vertices. Since $a, b \in B_H(v, \Delta \cdot d)$, there is a path $P(a, b)$ connecting a to b in H of length at most $2\Delta \cdot d = d' \leq \tilde{d}_1$. We then let $\mathcal{P}(A, B) = \{P(a, b) \mid (a, b) \in M\}$ be the resulting collection of paths, that routes every vertex of A to a distinct vertex of B. The length of every path in $\mathcal{P}(A, B)$ is at most $\tilde{d}_1$, and the congestion caused by the paths in $\mathcal{P}(A, B)$ is at most $|\mathcal{P}(A, B)| \leq |V(H)| \leq N < \eta_1$.

The running time of the algorithm is dominated by the running time of Procedure ProcSeparate, which is bounded by $O(|E(H)| \cdot N^{64/\Delta}) \leq O(|E(H)| \cdot N^{\epsilon'}) < c|E(H)| \cdot N^6$, if c is large enough, since $\Delta = 64/\epsilon'$.

Step: $1 < j \leq h$

We now assume that we are given some integer $1 < j \leq \lceil 1/\epsilon \rceil$, such that the statement of Theorem 5.2 holds for $j-1$, and we prove the statement of the theorem for j. Recall that we are given as input a graph H with $|V(H)| = N^j$.

We partition the set $V(H)$ of vertices into N subsets $V_1, \ldots, V_N$, each of which contains exactly N^{j-1} vertices. The algorithm consists of two phases. In the first phase, we run the Distanced Matching Game in parallel on N graphs $H_1, \ldots, H_N$, where for all $1 \leq i \leq N$, $V(H_i) = V_i$. We will add edges to graphs $H_1, \ldots, H_N$ gradually via the Distanced Matching Game, while computing an embedding of all resulting edges into graph H, so that the resulting embedding paths have sufficiently low length and cause sufficiently low congestion. In this phase, we will either compute the required (δ_j, d)-distancing in H, or we will be able to successfully complete the Distanced Matching Game on a sufficiently large collection $\mathcal{H}' \subseteq \{H_1, \ldots, H_N\}$ of the graphs. In the latter case, for each such graph $H_i \in \mathcal{H}'$, we will also compute a level-$(j-1)$ Hierarchical Support Structure for H_i. If the outcome of the first phase is a (δ_j, d)-distancing for H, then we terminate the algorithm and return this distancing. Otherwise, we continue to the second phase. In the second phase we will exploit the graphs in $\mathcal{H}'$ to either compute a (δ_j, d)-distancing in graph H, or to compute a subset $\mathcal{H}'' \subseteq \mathcal{H}'$ of r graphs, so that, if we let $S(H) = \bigcup_{H_i \in \mathcal{H}''} S(H_i)$, then graph H is $(\delta_j, \tilde{d}_j)$-well-connected with respect to the set $S(H)$ of vertices. We now describe each of the two phases in turn.

6.1 Phase 1: Construction of Smaller Well-Connected Graphs

In this phase, we gradually construct a collection $\mathcal{H} = \{H_1, \ldots, H_N\}$ of graphs, over the course of at most $N^{64\epsilon^2}$ iterations, by running the Distanced Matching Game over these graphs in parallel with parameters $\delta = \delta_{j-1}$ and $d = 2^{32/\epsilon^4}$. Initially, for all $1 \leq i \leq N$, we let $V(H_i) = V_i$ and $E(H_i) = \emptyset$. In every iteration $1 \leq q \leq N$, we will compute, for all $1 \leq i \leq N$, a partial matching E_i^q over the vertices of V_i. We will ensure that either $E_i^q = \emptyset$, or $|E_i^q| \geq N^{(j-1)(1-\delta_{j-1})}/8$. The edges of E_i^q are then added to graph H_i. Additionally, in the qth iteration, we will compute an embedding $\mathcal{P}^q$ of all edges in $\bigcup_{i=1}^N E_i^q$ into H. We now describe a single iteration q.

6.1.1 Description of Iteration q

Using the induction hypothesis, we apply the algorithm from Theorem 5.2 to each of the graphs $H_i \in \mathcal{H}$, with parameter $(j-1)$. We denote by $\mathcal{H}^1 \subseteq \mathcal{H}$ the collection of all graphs H_i, for which the algorithm returned a (δ_{j-1}, d)-distancing in H_i, and we denote by $\mathcal{H}^2 = \mathcal{H} \setminus \mathcal{H}^1$ all remaining graphs. Recall that, for each graph $H_i \in \mathcal{H}^2$, the algorithm computes a level-$(j-1)$ Hierarchical Support Structure. We now consider two cases, depending on whether $|\mathcal{H}^1| > N^{1-\epsilon'}$.

Case 1: $|\mathcal{H}^1| > N^{1-\epsilon'}$. If this case happens, then we say that iteration q is *regular*. Recall that, for every graph $H_i \in \mathcal{H}^1$, the algorithm from Theorem 5.2 returned a (δ_{j-1}, d)- distancing (A_i, B_i, E_i'), where $|A_i| = |B_i| \geq |V(H_i)|^{1-\delta_{j-1}} = N^{(j-1)(1-\delta_{j-1})}$, and $|E_i'| \leq \frac{|A_i|}{N^{\epsilon'}}$.

Let $z' = \left\lceil \frac{2 \log N}{\epsilon} \right\rceil$. We further partition the collection $\mathcal{H}^1$ of graphs into subsets $\mathcal{H}_0^1, \ldots, \mathcal{H}_{z'}^1$, where for all $0 \leq z \leq z'$, class $\mathcal{H}_z^1$ contains all graphs $H_i \in \mathcal{H}^1$, for which $\frac{N^{j-1}}{2^{z+1}} \leq |A_i| < \frac{N^{j-1}}{2^z}$ holds. Clearly, there must be an index $0 \leq z \leq z'$, such that:

$$|\mathcal{H}^1_z| \geq \frac{|\mathcal{H}^1| \cdot \epsilon}{4 \log N} \geq \frac{N^{1-\epsilon'} \cdot \epsilon}{4 \log N} \geq N^{1-2\epsilon'}. \tag{6}$$

(since $\frac{N^{\epsilon'}}{\log N} \geq 2^{128/\epsilon^5}$ from the statement of Theorem 5.2).

From now on we fix this index z. Since, for all $H_i \in \mathcal{H}^1$, $|A_i| \geq N^{(j-1)(1-\delta_{j-1})} \geq N^{j-j\delta_{j-1}-1}$, we get that:

$$2^z \leq N^{j\delta_{j-1}}. \tag{7}$$

For convenience, we denote by $I \subseteq \{1, \ldots, N\}$ the set of all indices i with $H_i \in \mathcal{H}^1_z$. Next, we apply Algorithm ProcPathPeel from Lemma 3.12, to graph H, collections of subsets $\{(A_i, B_i)\}_{i \in I}$ of its vertices, length parameter d', and congestion parameter $\eta = N^{8\epsilon^3}$. Consider the resulting collections $\{\mathcal{P}^q_i\}_{i \in I}$ of paths. Recall that, from Lemma 3.12, for all $i \in I$, every path in the corresponding set $\mathcal{P}^q_i$ has length at most d', and it connects a vertex of A_i to a vertex of B_i, so that the endpoints of all paths in $\mathcal{P}_i$ are distinct. Therefore, we can use the paths in $\mathcal{P}^q_i$ in order to define a partial matching M^q_i between vertices of A_i and vertices of B_i: a pair $(a, b) \in A_i \times B_i$ of vertices is added to M^q_i iff some path of $\mathcal{P}^q_i$ has endpoints a, b. We denote by $A'_i \subseteq A_i$ and $B'_i \subseteq B_i$ the subsets of vertices of A_i and B_i respectively, that do not serve as endpoints of the paths in $\mathcal{P}^q_i$. Let $E^* \subseteq E(H)$ be the set of all edges e that participate in exactly η paths of $\bigcup_{i \in I} \mathcal{P}^q_i$. Recall that Lemma 3.12 further guarantees that the paths of $\bigcup_{i \in I} \mathcal{P}^q_i$ cause congestion at most η in H. Moreover, if we consider graph $H \setminus E^*$, then for all $i \in I$, the length of a shortest path connecting a vertex of A'_i to a vertex of B'_i is greater than d'.

Running time of Algorithm ProcPathPeel is $O(|E(H)| \cdot (\eta + jNd' \log N)) \leq O(|E(H)| \cdot (N^{8\epsilon^3} + jNd' \log N)) \leq O(|E(H)| \cdot jNd' \log N)$, since $|\mathcal{H}| = N$.

We say that a graph $H_i \in \mathcal{H}^1_z$ is *successful* if $|\mathcal{P}^q_i| \geq |A_i|/4$, and it is unsuccessful otherwise. We let $\mathcal{G}^s \subseteq \mathcal{H}^1_z$ be the collection of all successful graphs, and $\mathcal{G}^u = \mathcal{H}^1_z \setminus \mathcal{G}^s$ the collection of all unsuccessful graphs. For convenience, we also partition the collection I of indices into a set I^s containing all indices $i \in I$ where $H_i \in \mathcal{G}^s$ and $I^u = I \setminus I^s$. We consider again two cases, depending on whether $|\mathcal{G}^s| \geq |\mathcal{H}^1_z|/2$.

Case 1a: $|\mathcal{G}^s| \geq |\mathcal{H}^1_z|/2$. If Case 1a happens then we say that iteration q is *successful*. Consider some index $i \in I^s$. Recall that the matching M^q_i that we have defined contains at least $|A_i|/4$ pairs of vertices (that we refer to as edges), while $|E'_i| \leq |A_i|/16$ by the definition of distancing. We discard from M^q_i all edges that lie in E'_i; note that $|M^q_i| \geq |A_i|/8$ continues to hold. We add the edges of the resulting matching M^q_i to graph H_i, and we say that graph H_i *received a matching* in iteration q. We view this matching as the response of the matching player in the Distanced Matching Game. We let $\mathcal{P}^q = \bigcup_{i \in I^s} \mathcal{P}^q_i$. Notice that the paths in $\mathcal{P}^q$ can be viewed as an embedding of the set $\bigcup_{i \in I^s} M^q_i$ of edges into graph H. The length of each path is at most d', and the congestion of the embedding is at most η. Observe that, if an iteration is successful, then the number of graphs in $\mathcal{H}$ that receive matchings is at least $|\mathcal{G}^s| \geq |\mathcal{H}^1_z|/2 \geq N^{1-2\epsilon'}/2$ (from Equation 6). We terminate the current iteration, and proceed to the next iteration.

Case 1b: $|\mathcal{G}^s| < |\mathcal{H}^1_z|/2$. If Case 1b happens, then we say that the current iteration is *unsuccessful*. In this case, we will construct a (δ_j, d)-distancing (A, B, E^*) in graph H, where E^* is the set of edges that we have defined above. We will ensure that $|E^*| \leq |A|/N^{j\epsilon^4}$. The current iteration then becomes the last iteration of the algorithm, and we return (A, B, E^*) as its output. We need the following simple observation bounding $|E^*|$:

Observation 6.1 $|E^*| \leq \frac{N^j \cdot d'}{\eta \cdot 2^z}$.

Proof: Note that for every graph $H_i \in \mathcal{H}^1_z$, $|\mathcal{P}^q_i| \leq |A_i| \leq \frac{N^{j-1}}{2^z}$. Since the length of each such path is at most d', the total number of edges on all paths in $\bigcup_{i \in I} \mathcal{P}^q_i \leq d' \cdot |\mathcal{H}^1_z| \cdot \frac{N^{j-1}}{2^z} \leq \frac{N^j \cdot d'}{2^z}$ (as $|\mathcal{H}^1_z \leq N$). Since set E^* only contains edges that lie on η path of $\mathcal{P}^q$, the observation follows. □

Recall that, if $H_i \in \mathcal{G}^u$, then $|A'_i| \geq \frac{|A_i|}{2} \geq \frac{N^{j-1}}{2^{z+2}}$. Recall also that $|B'_i| = |A'_i|$, and $\mathsf{dist}_{H \setminus E^*}(A'_i, B'_i) > d'$. We denote $T = \bigcup_{i \in I^u}(A'_i \cup B'_i)$, and we call the vertices of T *terminals*. Since we have assumed that $|\mathcal{G}^s| < |\mathcal{H}^1_z|/2$, we get that:

$$|T| \geq |\mathcal{G}^u| \cdot \frac{N^{j-1}}{2^{z+1}} \geq |\mathcal{H}^1_z| \cdot \frac{N^{j-1}}{2^{z+2}} \geq \frac{N^{j-2\epsilon'}}{2^{z+2}}. \tag{8}$$

(we have used Equation 6.)

We need the following simple observation.

Observation 6.2 *For every terminal $t \in T$, at most $|T|/2$ terminals may lie in $B_{H \setminus E^*}(t, d'/2)$.*

Proof: Consider any terminal $t \in T$, and denote $B^t = B_{H \setminus E^*}(t, d'/2)$. Recall that for all $H_{i'} \in \mathcal{G}^u$, $\mathsf{dist}_{H \setminus E^*}(A'_{i'}, B'_{i'}) > d'$. Therefore, B^t may not contain a vertex of $A'_{i'}$ and a vertex of $B'_{i'}$. We conclude that, for every terminal $t \in T$, $|B^t \cap T| \leq |T|/2$. □

We apply Algorithm ProcSeparate from Lemma 3.10 to graph $G = H \setminus E^*$, the set T of terminals, distance parameters d and Δ that remain unchanged, and $\alpha = 1/2$. The running time of the algorithm is $O(|E(H)| \cdot |V(H)|^{64/\Delta}) \leq O(|E(H)| \cdot N^{j\epsilon'})$.

From Observation 6.2, the algorithm may not return a terminal $t \in T$ with $|B_{H \setminus E^*}(t, \Delta \cdot d) \cap T| = |B_{H \setminus E^*}(t, d'/2) \cap T| > \alpha \cdot |T|$. Therefore, it must compute two subsets T_1, T_2 of terminals, with $|T_1| = |T_2|$, such that for every pair $t \in T_1, t' \in T_2$ of terminals, $\mathsf{dist}_{H \setminus E^*}(t, t') \geq d$. Moreover:

$$|T_1| \geq \frac{|T|^{1-64/\Delta}}{3} \geq \frac{|T|^{1-\epsilon'}}{3} \geq \frac{N^{(j-2\epsilon')(1-\epsilon')}}{3 \cdot 2^{(z+2)(1-\epsilon')}} \geq \frac{N^{j-2j\epsilon'}}{2^{z+4}}$$

(we have used the fact that $\Delta = 64/\epsilon'$, and Equation 8).

Recall that, from Observation 6.1, $|E^*| \leq \frac{N^j \cdot d'}{\eta \cdot 2^z}$. Recall also that $\eta = N^{8\epsilon^3}$, $j \leq \lceil 1/\epsilon \rceil$, and $d' \leq 2^{64/\epsilon^4 \leq N^{\epsilon'}}$ from Inequality 5, and since $\frac{N^{\epsilon'}}{\log N} \geq 2^{128/\epsilon^5}$ from the statement of Theorem 5.2. Therefore, we get that:

$$|E^*| \leq \frac{N^j}{2^z} \cdot \frac{N^{\epsilon^4}}{N^{8\epsilon^3}} \leq \frac{N^j}{2^{z+4}} \cdot \frac{1}{N^{6\epsilon^3}} \leq \frac{N^j}{2^{z+4}} \cdot \frac{1}{N^{3\epsilon^4 \cdot \lceil 1/\epsilon \rceil}} \leq \frac{N^j}{2^{z+4}} \cdot \frac{1}{N^{3\epsilon' j}} \leq \frac{|T_1|}{N^{j\epsilon'}}.$$

Lastly, recall that we have shown in Inequality 7, that $2^z \leq N^{j\delta_{j-1}}$. Therefore, we get that:

$$|T_1| \geq \frac{N^j}{16N^{j(2\epsilon' + \delta_{j-1})}} \geq \frac{N^j}{16N^{j(4(j-1)\epsilon' + 2\epsilon')}} \geq \frac{N^j}{N^{j(4j\epsilon')}} = N^{j(1-\delta_j)}.$$

We conclude that (T_1, T_2, E^*) is a (δ_j, d)-distancing, with $|E^*| \leq |T_1|/N^{j\epsilon'}$ as required. We return this distancing and terminate the algorithm.

Case 2: $|\mathcal{H}^1| \leq N^{1-\epsilon'}$**.** If this case happens, then we say that the current iteration is *irregular*. In this case, we terminate Phase 1. The outcome of the phase is the collection $\mathcal{H}^2 \subseteq \mathcal{H}$ of at least $N - N^{1-\epsilon'}$ graphs. Recall that, for each graph $H_i \in \mathcal{H}^2$, we computed a level-$(j-1)$ Hierarchical Support Structure, that includes a subset $S(H_i) \subseteq V(H_i)$ of vertices, such that H_i is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected with respect to $S(H_i)$. Let $H' = \bigcup_{H_i \in \mathcal{H}^2} H_i$. Notice that the sets $\mathcal{P}^q$ of paths that are computed in each iteration provide an embedding of graph H' into H. The length of each resulting path is bounded by d'. We use the following observation in order to both bound the congestion of this embedding, and the running time of the algorithm.

Observation 6.3 *At the end of Phase 1, for each graph $H_i \in \mathcal{H}$, $|E(H_i)| \leq N^{j-1+32\epsilon^2}$. The number of iterations in Phase 1 is at most $N^{64\epsilon^2}$.*

Proof: Recall first that $\delta_{j-1} = 4(j-1)\epsilon' \geq 4\epsilon^4$, while $d = 2^{32/\epsilon^4}$. Therefore, $d \geq 2^{4/\delta_{j-1}}$ holds. We can then view our algorithm from Phase 1 as running the Distanced Matching Game simultaneously over the graphs in $\mathcal{H}$. Note that for every graph $H_i \in \mathcal{H}$:

$$|V(H_i)|^{\delta_{j-1}} = N^{(j-1)\delta_{j-1}} \geq N^{4\epsilon'} \geq \frac{2^{128}\log N}{\epsilon^{11}} \geq \frac{2^{124}\log N}{16(\epsilon')^2\epsilon^3} \geq \frac{2^{14}(j-1)\log N}{\delta_{j-1}^2} = \frac{2^{14}\log(|V(H_i)|)}{\delta_{j-1}^2},$$

since $\delta_{j-1} = 4(j-1)\epsilon'$, $2 \leq j \leq \lceil 1/\epsilon \rceil$, and $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^5}$ from the statement of Theorem 5.2. We conclude that $|V(H_i)|^{\delta_{j-1}} \geq \frac{2^{14}\log(|V(H_i)|)}{\delta_{j-1}^2}$ holds, satisfying the conditions of Theorem 4.2. From Theorem 4.2, the number of iterations in a Distanced Matching Game in a single graph $H_i \in \mathcal{H}$ is bounded by:

$$|V(H_i)|^{8\delta_{j-1}} \leq N^{(j-1)\cdot 8\delta_{j-1}} \leq N^{32(j-1)^2\epsilon'} \leq N^{32\epsilon^2},$$

since $\epsilon' = \epsilon^4$, $\delta_{j-1} = 4(j-1)\epsilon'$, and $j \leq \lceil 1/\epsilon \rceil$. Therefore, a graph $H_i \in \mathcal{H}$ may receive a matching in at most $N^{32\epsilon^2}$ iterations, and the cardinality of each such matching is at most $|V(H_i)|/2 \leq N^{j-1}/2$. We conclude that at the end of Phase 1, for each graph $H_i \in \mathcal{H}$, $|E(H_i)| \leq N^{j-1} \cdot N^{32\epsilon^2}$.

Next, we bound the number of iterations in Phase 1. Note that at most one iteration of Phase 1 may be irregular, and at most one iteration may be regular and unsuccessful. It is now enough to bound the number of regular and successful iterations. In every regular and successful iteration, at least $N^{1-2\epsilon'}/2$ graphs in $\mathcal{H}$ receive matchings. Therefore, every regular and successful iteration of Phase 1 results in the completion of a single iteration of the Distanced Matching Game in at least $N^{1-2\epsilon'}/2$ graphs of $\mathcal{H}$. At the same time, the number of pairs (i,q), where graph $H_i \in \mathcal{H}$ receives a matching in iteration q must be bounded by $|\mathcal{H}| \cdot N^{32\epsilon^2} \leq N^{1+32\epsilon^2}$. Since in every regular and successful iteration at least $N^{1-2\epsilon'}/2$ graphs in $\mathcal{H}$ receive matchings, we get that the number of iterations is bounded by $2N^{32\epsilon^2+2\epsilon'} \leq N^{64\epsilon^2}$. □

Since every set $\mathcal{P}^q$ of paths causes congestion at most $\eta = N^{8\epsilon^3}$, and the number of iterations is bounded by $N^{64\epsilon^2}$, we obtain an embedding of graph H' into H via paths of length at most $d' \leq 2^{64/\epsilon^4}$ (from Equation 5), that cause total congestion at most $N^{64\epsilon^2} \cdot N^{8\epsilon^3} \leq N^{128\epsilon^2}$. We denote this embedding by $\mathcal{P}$.

Note that the Phase 1 can either terminate with a regular unsuccessful iteration, in which case we terminate the algorithm and return the resulting distancing for graph H, or with an irregular iteration, in which case we proceed to Phase 2 of the algorithm. Before we describe Phase 2 of the algorithm, we analyze the running time of Phase 1.

Running Time Analysis of Phase 1.

We start by bounding the running time of a single iteration. Over the course of the iteration, we apply the algorithm from Theorem 5.2 to each of the graphs $H_i \in \mathcal{H}$, with parameter $(j-1)$. Recall that, from Observation 6.3, for each graph $H_i \in \mathcal{H}$, $|E(H_i)| \leq N^{j-1+32\epsilon^2}$. From the induction hypothesis, the running time of the algorithm from Theorem 5.2 on a single graph $H_i \in \mathcal{H}$ is bounded by:

$$\begin{aligned} c(j-1) \cdot N^{(j-1)(1+64\epsilon^2)+7} &+ c|E(H_i)| \cdot N^6 \\ &\leq c(j-1) \cdot N^{j+64(j-1)\epsilon^2+6} + cN^{j-1+32\epsilon^2} \cdot N^6 \\ &= c(j-1) \cdot N^{j+64(j-1)\epsilon^2+6} + cN^{j+5+32\epsilon^2}. \\ &\leq cj \cdot N^{j+64(j-1)\epsilon^2+6}. \end{aligned}$$

Since $|\mathcal{H}| = N$, the running time of this part of the algorithm is bounded by:

$$cj \cdot N^{j+64(j-1)\epsilon^2+7}$$

If the iteration is regular, then we apply Algorithm ProcPathPeel from Lemma 3.12, to graph H, collections of subsets $\{(A_i, B_i)\}_{i \in I}$ of its vertices, length parameter d', and congestion parameter $\eta = N^{8\epsilon^3}$. As observed above, the running time of Algorithm ProcPathPeel is $O(|E(H)| \cdot jNd' \log N) = O(|E(H)| \cdot N \cdot 2^{64/\epsilon^4} \cdot \log N/\epsilon) \leq O(|E(H)| \cdot N^2)$, since $d' \leq 2^{64/\epsilon^4}$ from Equation 5 and $N^{\epsilon'} > 2^{64/\epsilon^5} \cdot \log N$ from the statement of Theorem 5.2.

If the iteration is regular and unsuccessful, then we apply Algorithm ProcSeparate from Lemma 3.10, whose running time, as observed above, is bounded by $O(|E(H)| \cdot N^{j\epsilon'}) \leq O(|E(H)| \cdot N^{2\epsilon^3})$, as $j \leq \lceil 1/\epsilon \rceil$ and $\epsilon' = \epsilon^4$.

Overall, the running time of a single iteration is bounded by:

$$cj \cdot N^{j+64(j-1)\epsilon^2+7} + O\left(|E(H)| \cdot N^2\right).$$

Since, from Observation 6.3, the number of iterations in Phase 1 is at most $N^{64\epsilon^2}$, we get that the total running time of Phase 1 is bounded by:

$$cj \cdot N^{j+64j\epsilon^2+7} + O\left(|E(H)| \cdot N^3\right).$$

6.2 Phase 2: Distancing or Well-Connectedness

The starting point of Phase 2 is the collection $\mathcal{H}^2 \subseteq \mathcal{H}$ of at least $N - N^{1-\epsilon'}$ graphs that was computed in Phase 1. Recall that, for each graph $H_i \in \mathcal{H}_2$, we computed a level-$(j-1)$ Hierarchical Support Structure, that includes a subset $S(H_i) \subseteq V(H_i)$ of vertices, such that H_i is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected with respect to $S(H_i)$. Additionally, we have computed an embedding $\mathcal{P}$ of graph $H' = \bigcup_{H_i \in \mathcal{H}^2} H_i$, so that the length of each path in $\mathcal{P}$ is at most $d' \leq 2^{64/\epsilon^4}$, and the paths in $\mathcal{P}$ cause congestion at most $N^{128\epsilon^2}$.

In this phase we will either compute a subset $\mathcal{H}' \subseteq \mathcal{H}^2$ of r graphs, such that graph H is $(\eta_j, \tilde{d}_j)$-well-connected with respect to the set $S(H) = \bigcup_{H_i \in \mathcal{H}'} S(H_i)$ of vertices; or we compute a (δ_j, d)-distancing (A, B, E') in graph H, with $|E'| \leq \frac{|A|}{N^{j\epsilon^4}}$.

We consider every pair $H_i, H_{i'} \in \mathcal{H}^2$ of graphs, with $i < i'$ one by one. When the pair $(H_i, H_{i'})$ of graphs is considered, we apply Procedure ProcPathPeel from Lemma 3.12 to graph H, and two sets $A_1 = S(H_i), B_1 = S(H_{i'})$ of its vertices, with distance parameter d' and congestion parameter $\eta = N^4$. Recall that the running time of the procedure is $O(|E(H)|(N^4 + jd' \log N)) \leq O(|E(H)| \cdot N^4)$. We denote by $\mathcal{Q}_{i,i'}$ the set of paths that the algorithm returns, and by $E'_{i,i'}$ the set of all edges of H that participate in N^4 paths of $\mathcal{Q}_{i,i'}$. We also denote by $A'_{i,i'} \subseteq S(H_i)$ and $B'_{i,i'} \subseteq S(H_{i'})$ the sets of vertices that do not serve as endpoints of paths in $\mathcal{Q}_{i,i'}$. Recall that the paths in $\mathcal{Q}_{i,i'}$ have length at most d' each, and they cause congestion at most N^4. Every path in $\mathcal{Q}_{i,i'}$ connects a vertex of $S(H_i)$ to a vertex of $S(H_{i'})$, and every vertex of $S(H_i) \cup S(H_{i'})$ may serve as an endpoint of at most one path in $\mathcal{Q}_{i,i'}$. Moreover, the length of the shortest path in $H \setminus E'_{i,i'}$ connecting a vertex of $A'_{i,i'}$ to a vertex of $B'_{i,i'}$ is greater than d'. Observe also that, since $|\mathcal{Q}_{i,i'}| \leq |S(H_i)| \leq |V(H_i)| = N^{j-1}$, and since the length of every path in $\mathcal{Q}_{i,i'}$ is at most d', we get that $\sum_{Q \in \mathcal{Q}_{i,i'}} |E(Q)| \leq d' \cdot N^{j-1}$. Since $E'_{i,i'}$ only contains edges that participate in N^4 paths in $\mathcal{Q}$, we get that $|E'_{i,i'}| \leq d' \cdot N^{j-5}$.

Let E' be the union of all sets $E'_{i,i'}$ of edges, over all pairs $H_i, H_{i'} \in \mathcal{H}^2$ of graphs with $i < i'$. Clearly, $|E'| \leq d' \cdot N^{j-3}$. Moreover, for every pair $H_i, H_{i'} \in \mathcal{H}^2$ of graphs with $i < i'$, the length of the shortest path in $H \setminus E'$ connecting a vertex of $A'_{i,i'}$ to a vertex of $B'_{i,i'}$ is greater than d'.

Next, we apply procedure ProcSeparate from Lemma 3.10 to graph $\tilde{H} = H \setminus E'$, with the set $T = V(H)$ of terminal vertices, distance parameters d and Δ that remain unchanged, and parameter $\alpha = 1 - \frac{1}{8N^{\epsilon'}}$. Recall that the running time of the algorithm is $O(|E(H)| \cdot N^{64j/\Delta}) \leq O(|E(H)| \cdot N^{2\epsilon^3})$ (since $\Delta = 64/\epsilon'$, $j \leq \lceil 1/\epsilon \rceil$, and $\epsilon' = \epsilon^4$).

We now consider two cases. The first case is that Procedure ProcSeparate returns two subsets $A, B \subseteq V(H)$ of vertices, such that $|A| = |B|$, and for every pair $v \in A, u \in B$ of vertices, $\mathsf{dist}_{\tilde{H}}(u, v) \geq d$. Recall that in this case, the algorithm also ensures that:

$$\begin{aligned} |A| &\geq |V(H)|^{1-64/\Delta} \cdot \min\left\{(1-\alpha), \frac{1}{3}\right\} \\ &\geq N^{j(1-\epsilon')} \cdot \frac{1}{8N^{\epsilon'}} \\ &\geq N^{j(1-\epsilon')-2\epsilon'} \\ &\geq N^{j(1-4j\epsilon')} \\ &= |V(H)|^{1-\delta_j}. \end{aligned} \tag{9}$$

(We have used the fact that $\Delta = 64/\epsilon'$ and $\delta_j = 4j\epsilon'$).

Recall that:

$$|E'| \leq d' \cdot N^{j-3} \leq N^{j-3+\epsilon'}.$$

(since $d' \leq 2^{64/\epsilon^3}$ from Equation 5 and $N^{\epsilon^4} \geq 2^{128/\epsilon^5}$ from the statement of Theorem 5.2.) Since, from Equation 9, $|A| \geq N^{j-j\epsilon'-2\epsilon'} \geq N^{j-3\epsilon^3}$ (as $\epsilon' = \epsilon^4$), we get that $|E'| \leq |A|/N^{j\epsilon^4}$.

We conclude that (A, B, E') is a valid (δ_j, d)-distancing in graph H, with $|E'| \leq |A|/N^{j\epsilon^4}$. We return this distancing and terminate the algorithm.

From now on we assume that Procedure ProcSeparate computed a vertex $v \in V(H)$, such that $|B_{\tilde{H}}(v, \Delta \cdot d)| > \alpha \cdot |V(H)| = N^j \cdot \left(1 - \frac{1}{8N^{\epsilon'}}\right)$. For convenience, we denote $B^* = B_{\tilde{H}}(v, \Delta \cdot d)$.

In this case, we construct a set $\mathcal{H}' \subseteq \mathcal{H}^2$ of graphs as follows: we add graph H_i to $\mathcal{H}'$ iff $|B^* \cap V(H_i)| \geq \frac{7|V(H_i)|}{8}$. We need the following observation.

Observation 6.4 $|\mathcal{H}'| \geq N - 2N^{1-\epsilon'}$.

Proof: Recall that $|\mathcal{H}^2| \geq N - N^{1-\epsilon'}$. Notice that, if $H_i \in \mathcal{H}^2 \setminus \mathcal{H}'$, then $|V(H_i) \setminus B^*| \geq |V(H_i)|/8 = N^{j-1}/8$. Since $|V(H) \setminus B^*| \leq N^{j-\epsilon'}/8$, we get that $|\mathcal{H}^2 \setminus \mathcal{H}'| \leq \frac{N^{j-\epsilon'}/8}{N^{j-1}/8} = N^{1-\epsilon'}$. Therefore, $|\mathcal{H}'| \geq |\mathcal{H}^2| - N^{1-\epsilon'} \geq N - 2N^{1-\epsilon'}$. □

We discard arbitrary graphs from $\mathcal{H}'$, until $|\mathcal{H}'| = N - \left\lceil 2N^{1-\epsilon'} \right\rceil$ holds. Let $S(H) = \bigcup_{H_i \in \mathcal{H}'} S(H_i)$. Note that we have now obtained a level-j Hierarchical Support Structure for graph H, whose associated collection of graphs is $\mathcal{H}'$, with $|\mathcal{H}'| = N - \left\lceil 2N^{1-\epsilon'} \right\rceil = r$. We use the embedding $\mathcal{P}$ of the graph $\bigcup_{H_i \in \mathcal{H}^2} H_i$ that we have computed in Phase 1. By discarding paths that are no longer needed from $\mathcal{P}$, we obtain an embedding $\mathcal{P}$ of graph $\bigcup_{H_i \in \mathcal{H}'} H_i$ into H, such that every path in the embedding has length at most $d' \leq 2^{64/\epsilon^4}$, and the paths in $\mathcal{P}$ cause congestion at most N^{128/ϵ^3}. We prove the following lemma in Section 6.3.

Lemma 6.5 *Graph H is $(\eta_j, \tilde{d}_j)$-well-connected with respect to $S(H)$.*

In order to complete the proof of Theorem 5.2, it is now enough to show that the running time of the algorithm is suitably bounded.

Running Time Analysis

The algorithm performs at most N^2 calls to Procedure ProcPathPeel. As observed above, the running time of each such call is at most $O(|E(H)| \cdot N^4)$. The running time of Procedure ProcSeparate, as shown above, is at most $O(|E(H)| \cdot N^{2\epsilon^3})$. Overall, the running time of Phase 2 is $O(|E(H)| \cdot N^6)$.

Altogether, the running time of the whole algorithm is bounded by:

$$cj \cdot N^{j+64j\epsilon^2+7} + O\left(|E(H)| \cdot N^6\right) \leq cj \cdot N^{j(1+64\epsilon^2)+7} + c|E(H)| \cdot N^6),$$

if c is a large enough constant.

6.3 Proof of Lemma 6.5

We will use the following simple observation.

Observation 6.6 *For every pair $H_i, H_{i'} \in \mathcal{H}'$ with $i < i'$, $|\mathcal{Q}_{i,i'}| \geq N^{j-1}/4$. (Here, $\mathcal{Q}_{i,i'}$ is the set of paths that was computed in Phase 2 of the algorithm.)*

Proof: Recall that, from Claim 5.1, $|V(H_i) \setminus S(H_i)| \leq |V(H_i)| \cdot \frac{4(j-1)}{N^{\epsilon^4}} < \frac{|V(H_i)|}{64}$ (since $j \leq \lceil \frac{1}{\epsilon} \rceil$). Therefore, $|S(H_i)| \geq \frac{63|V(H_i)|}{64} = \frac{63 \cdot N^{j-1}}{64}$, and similarly, $|S(H_{i'})| \geq \frac{63 \cdot N^{j-1}}{64}$.

Assume now for contradiction that $|\mathcal{Q}_{i,i'}| < \frac{N^{j-1}}{4}$. Recall that we have defined sets $A'_{i,i'} \subseteq S(H_i)$ and $B'_{i,i'} \subseteq S(H_{i'})$ of vertices that do not serve as endpoints of paths in $\mathcal{Q}_{i,i'}$. If $|\mathcal{Q}_{i,i'}| < \frac{N^{j-1}}{4}$, then:

$$|A'_{i,i'}| \geq |S(H_i)| - |\mathcal{Q}_{i,i'}| \geq \frac{63 \cdot N^{j-1}}{64} - \frac{N^{j-1}}{4} > \frac{2 \cdot N^{j-1}}{3}.$$

Similarly, $|B'_{i,i'}| > \frac{2 \cdot N^{j-1}}{3}$. Since H_i was added to $\mathcal{H}'$, $|B^* \cap V(H_i)| \geq \frac{7 \cdot N^{j-1}}{8}$. So at least one vertex $u \in A'_{i,i'}$ must lie in B^*. For similar reasons, at least one vertex $u' \in B'_{i,i'}$ must lie in B^*. From the definition of B^*, $\mathsf{dist}_{\tilde{H}}(u, u') \leq 2\Delta \cdot d = d'$. Recall however that $\tilde{H} = H \setminus E'$, and $E'_{i,i'} \subseteq E'$. Procedure $\mathsf{ProcPathPeel}$ guarantees that the shortest path connecting u to u' in graph $H \setminus E'_{i,i'}$ has length greater than d'. So $\mathsf{dist}_{\tilde{H}}(u, u') > d'$ must hold, contradicting our previous claim that $\mathsf{dist}_{\tilde{H}}(u, u') \leq d'$. □

We now turn to the proof of Lemma 6.5. We assume that we are given two disjoint equal-cardinality subsets A, B of vertices of $S(H)$. Our goal is to prove that there exists a set $\mathcal{P}^*$ of paths in graph H, routing every vertex of A to a distinct vertex of B, such that the paths in $\mathcal{P}^*$ cause congestion at most η_j in H, and the length of every path is at most $\tilde{d}_j$. We will prove that such a set of paths exists by exploiting the fact that, every graph $H_i \in \mathcal{H}'$ is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected with respect to the set $S(H_i)$ of its vertices, together with the embedding of these graphs into H, and the sets $\mathcal{Q}_{i,i'}$ of paths that we have computed in Phase 2 of the algorithm for every pair $H_i, H_{i'} \in \mathcal{H}'$ of graphs with $i < i'$.

The remainder of the proof of Lemma 6.5 consists of three steps. In the first step, we route some pairs in $A \times B$ within the graphs $H_i \in \mathcal{H}'$. After the completion of this step, for every graph $H_i \in \mathcal{H}'$, either all vertices of $S(H_i)$ that remain to be routed lie in A, or all such vertices lie in B. In the remaining two steps we complete the routing of these remaining vertices. Specifically, in Step 2 we define a "meta-graph" $\hat{G}$, whose vertices represent the graphs $H_i \in \mathcal{H}'$, with weights on its edges. Intuitively, if an edge connecting two vertices that represent graphs H_i and $H_{i'}$ has weight $w(e)$, then we intend to construct $w(e)$ paths that connect vertices of $S(H_i) \cap A$ to vertices of $S(H_{i'}) \cap B$. In this step, we also perform a "global routing": for every pair $H_i, H_{i'} \in \mathcal{H}'$ of graphs, whose corresponding edge in $\hat{G}$ has weight w, we connect vertices of $S(H_i)$ to vertices of $S(H_{i'})$ with w paths. In the third and the final step, we complete the construction of the set $\mathcal{P}^*$ of paths by using "local routing", in which some pairs of vertex subsets are routed within each graph $H_i \in \mathcal{H}'$. We now describe each of the three steps in turn.

Step 1: Initial Routing within the Graphs of $\mathcal{H}'$

We process every graph $H_i \in \mathcal{H}'$ one by one. When graph H_i is processed, we denote $N_i^A = |A \cap S(H_i)|$ and $N_i^B = |B \cap S(H_i)|$. Denote $\beta_i = \min\left\{N_i^A, N_i^B\right\}$. Next, we select two arbitrary subsets $X_i \subseteq A \cap S(H_i)$ and $Y_i \subseteq B \cap S(H_i)$, each of which contains exactly β_i vertices. Since our algorithm ensures that graph H_i is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected with respect to $S(H_i)$, there exists a set $\mathcal{R}_i$ of paths in graph H_i, which is a one-to-one routing of vertices of X_i to vertices of Y_i. Every path in $\mathcal{R}_i$ has length at most $\tilde{d}_{j-1}$, and the paths in $\mathcal{R}_i$ cause congestion at most η_{j-1} in H_i.

Let $H' = \bigcup_{H_i \in \mathcal{H}'} H_i$. Recall that we have computed, in Phase 1 of the algorithm, an embedding $\mathcal{P}$ of H' into H, where every path in the embedding has length at most d', and the paths in $\mathcal{P}$ cause congestion at most $N^{128\epsilon^2}$ in H.

Consider now the set $\bigcup_{H_i \in \mathcal{H}'} \mathcal{R}_i$ of paths in graph H'. This set of paths defines a one-to-one routing of vertex set $X = \bigcup_{H_i \in \mathcal{H}'} X_i$ to vertex set $Y = \bigcup_{H_i \in \mathcal{H}'} Y_i$, where the length of every path is at most $\tilde{d}_{j-1}$, and the paths in $\mathcal{R}$ cause congestion at most η_{j-1} in H'. We now use the embedding $\mathcal{P}$ of H' into H in order to compute a set $\mathcal{P}'$ of paths in graph H, that route every vertex of X to a distinct vertex of Y, via the algorithm from Observation 3.1. We are then guaranteed that the length of every path in $\mathcal{P}'$ is at most $\tilde{d}_{j-1} \cdot d'$. Recall that $\tilde{d}_{j-1} = 2^{c(j-1)/\epsilon^4}$, while $d' < \frac{1}{4} \cdot 2^{c/\epsilon^4}$ from Inequality 5. Therefore, the length of every path in $\mathcal{P}'$ is bounded by:

$$\tilde{d}_{j-1} \cdot d' \leq 2^{c(j-1)/\epsilon^4} \cdot 2^{c/\epsilon^4} = 2^{cj/\epsilon^4} = \tilde{d}_j.$$

The algorithm from Observation 3.1 also guarantees that the congestion caused by the paths in $\mathcal{P}'$ in H is at most $\eta_{j-1} \cdot N^{128\epsilon^2}$. Since $\eta_{j-1} = N^{6+256(j-1)\epsilon^2}$, we get that the congestion caused by the paths in $\mathcal{P}'$ is bounded by:

$$\eta_{j-1} \cdot N^{128\epsilon^2} \leq N^{6+256(j-1)\epsilon^2} \cdot N^{128\epsilon^2} \leq \frac{N^{6+256j\epsilon^2}}{2} = \frac{\eta_j}{2}.$$

We have now obtained a set $\mathcal{P}'$ of paths in graph H, that routes every vertex of X to a distinct vertex of Y, so that the length of every path is at most $\tilde{d}_j$, and the congestion caused by the paths in $\mathcal{P}'$ is bounded by $\eta_j/2$.

We partition the graphs of $\mathcal{H}'$ into three subsets. Set $\mathcal{H}^N$ contains all graphs $H_i \in \mathcal{H}'$, in which $N_i^A = N_i^B$. We no longer need to route any vertices in such graphs, as for each such graph $\beta_i = N_i^A = N_i^B$; $X_i = A \cap S(H_i)$; and $Y_i = B \cap S(H_i)$ must hold. Set $\mathcal{H}^A$ contains graphs $H_i \in \mathcal{H}'$ with $N_i^A > N_i^B$. For each such graph H_i, we denote by $D(H_i) = N_i^A - N_i^B$, and by $X_i' = (A \cap S(H_i)) \setminus X_i$ – the set of vertices of $S(H_i)$ that remain to be routed. Clearly, $|X_i'| = D(H_i)$. Similarly, set $\mathcal{H}^B$ contains graphs $H_i \in \mathcal{H}'$ with $N_i^B > N_i^A$. For each such graph H_i, we denote by $D(H_i) = N_i^B - N_i^A$, and by $X_i' = (B \cap S(H_i)) \setminus Y_i$ – the set of vertices that remain to be routed. As before, $|X_i'| = D(H_i)$ holds. Notice also that, since $|A| = |B|$, $\sum_{H_i \in \mathcal{H}^A} D(H_i) = \sum_{H_i \in \mathcal{H}^B} D(H_i)$ must hold. Denote $\hat{A} = \bigcup_{H_i \in \mathcal{H}^A} X_i'$ and $\hat{B} = \bigcup_{H_i \in \mathcal{H}^B} X_i'$. It is now enough to prove the following lemma.

Lemma 6.7 *There is a set $\mathcal{P}''$ of paths in graph H, routing every vertex of $\hat{A}$ to a distinct vertex of $\hat{B}$, so that the length of every path in $\mathcal{P}''$ is at most $\tilde{d}_j$, and the paths in $\mathcal{P}''$ cause congestion at most $\eta_j/2$ in H.*

Indeed, by letting $\mathcal{P}^* = \mathcal{P}' \cup \mathcal{P}''$, we obtain a set of paths in graph H that defines a one-to-one routing of the set A of vertices to the set B of vertices via paths of length at most $\tilde{d}_j$, so that the paths in $\mathcal{P}^*$ cause congestion at most η_j. In order to complete the proof of Lemma 6.5, it is now enough to prove Lemma 6.7. We focus on the proof of Lemma 6.7 in the remainder of this section.

We will start by constructing a "meta-graph" representing the graphs of $\mathcal{H}^A \cup \mathcal{H}^B$, that will guide the construction of global routing.

Step 2: Meta-Graph and Global Routing

Abusing the notation, for simplicity, in the remainder of this proof we denote $\mathcal{H}^A = \{H_1, H_2, \ldots, H_q\}$, and for all $1 \leq i \leq q$, we denote $D(H_i)$ by D_i. We also denote $\mathcal{H}^B = \left\{H_1', H_2', \ldots, H_{q'}'\right\}$, and for $1 \leq i' \leq q'$, we denote $D(H_{i'}')$ by $D_{i'}'$. For all $1 \leq i \leq q$, we denote the set $X_i' \subseteq S(H_i)$ of D_i vertices that remains to be routed by Y_i, and for all $1 \leq i' \leq q'$, we denote the corresponding subset of D_i' vertices of $S(H_{i'}')$ by $Y_{i'}'$. We now define a *routing meta-graph*, that will be used in order to guide the construction of the paths in $\mathcal{P}^*$, and show that such a graph exists.

Routing Meta-Graph

We start by defining a routing meta-graph.

Definition 6.1 (Routing Meta-Graph) *A bipartite graph $\hat{G} = (U, U', \hat{E})$ with integral weights $w(e) \geq 0$ on its edges $e \in \hat{E}$ is a* routing meta-graph *if:*

- $U = \{v_1, \ldots, v_q\}$;
- $U' = \left\{v'_1, \ldots, v'_{q'}\right\}$;
- *for every vertex $v_i \in U$, $\sum_{e \in \delta_{\hat{G}}(v_i)} w(e) = D_i$; and*
- *for every vertex $v'_{i'} \in U'$, $\sum_{e \in \delta_{\hat{G}}(v'_{i'})} w(e) = D'_{i'}$.*

We refer to vertices of $\hat{G}$ as supernodes and edges of $\hat{G}$ as meta-edges.

We use the following claim to show that a routing meta-graph exists.

Claim 6.8 *There exists a routing meta-graph.*

Proof: We start with the graph $\hat{G} = (U, U', \hat{E})$, where $U = \{v_1, \ldots, v_q\}$, $U' = \left\{v'_1, \ldots, v'_{q'}\right\}$, and $\hat{E} = \emptyset$, and then iterate, as long as there exist indices $1 \leq i \leq q$ and $1 \leq i' \leq q'$, such that $D_i > 0$ and $D'_{i'} > 0$ holds.

In order to execute an iteration, we consider any pair of such indices (i, i'). Let $\hat{\Delta} = \min\left\{D_i, D'_{i'}\right\}$. We add an edge $(v_i, v'_{i'})$ to $\hat{E}$, whose weight is $\hat{\Delta}$, and we decrease D_i and $D'_{i'}$ by $\hat{\Delta}$. Once the algorithm terminates, since $\sum_{i=1}^{q} D_i = \sum_{i'=1}^{q'} D'_{i'}$, it is immediate to verify that the resulting graph $\hat{G}$ is a valid routing meta-graph. □

Global Routing

Consider some pair $v_i \in U$, $v'_{i'} \in U'$ of supernodes in graph $\hat{G}$. From Observation 6.6, there exists a collection of paths in graph H, that we denote, abusing the notation, by $\mathcal{Q}_{i,i'}$, such that the following hold:

- every path in $\mathcal{Q}_{i,i'}$ originates at a vertex of $S(H_i)$ and terminates at a vertex of $S(H'_{i'})$;
- every vertex of $S(H_i) \cup S(H'_{i'})$ is an endpoint of at most one path in $\mathcal{Q}_{i,i'}$;
- $|\mathcal{Q}_{i,i'}| = \left\lceil N^{j-1}/4 \right\rceil$;
- each path in $\mathcal{Q}_{i,i'}$ has length at most d'; and
- the paths in $\mathcal{Q}_{i,i'}$ cause congestion at most N^4 in H.

The set $\mathcal{Q}_{i,i'}$ of paths naturally defines a matching $M_{i,i'} \subseteq S(H_i) \times S(H'_{i'})$: we include a pair (x, y) of vertices in $M_{i,i'}$ if $x \in S(H_i)$, $y \in S(H'_{i'})$, and some path in $\mathcal{Q}_{i,i'}$ has x and y as its endpoints. Clearly, $|M_{i,i'}| = \left\lceil N^{j-1}/4 \right\rceil$. Notice that for every meta-edge $e = (v_i, v'_{i'})$ in graph $\hat{G}$, $w(e) \leq D_i \leq |S(H_i)| \leq N^{j-1}$ must hold. We will select, for every edge $e = (v_i, v'_{i'}) \in \hat{G}$, a multi-set $M'_{i,i'}$ of pairs $(x, y) \in M_{i,i'}$ of vertices, of cardinality $w(e)$. (We note that a pair $(x, y) \in M_{i,i'}$ of vertices may be added to $M'_{i,i'}$ multiple times). We will ensure that, for every supernode $v_i \in U$, a vertex $x \in S(H_i)$ may participate in at most four pairs in $\bigcup_{e=(v_i,v'_{i'}) \in \delta_{\hat{G}}(v_i)} M'_{i,i'}$, and the same holds for supernodes of

U'. For every meta-edge $e = (v_i, v'_{i'}) \in \hat{E}$, we will then use the paths of $\mathcal{Q}_{i,i'}$ whose endpoints lie in $M'_{i,i'}$ to define a *global routing*. Let $\mathcal{Q}^0$ denote the resulting collection (multiset) of all such paths. So for every meta-edge $(v_i, v'_{i'}) \in \hat{E}$, for every pair $(x, y) \in M'_{i,i'}$ of vertices, $\mathcal{Q}^0$ contains the path of $\mathcal{Q}_{i,i'}$ whose endpoints are x and y. If (x, y) appears multiple times in $M'_{i,i'}$, then $\mathcal{Q}^0$ contains multiple copies of this path. For every graph $\tilde{H} \in \mathcal{H}^A \cap \mathcal{H}^B$, for every vertex $x \in S(\tilde{H})$, we denote by $k(x)$ the number of paths of $\mathcal{Q}^0$, for which x serves as an endpoint. Our construction will guarantee that, for all $1 \leq i \leq q$, $\sum_{x \in S(H_i)} k(x) = D_i$, and similarly, for all $1 \leq i' \leq q'$, $\sum_{x \in S(H'_{i'})} k(x) = D'_{i'}$. As mentioned already, we will ensure that, for every graph $\tilde{H} \in \mathcal{H}'$, for every5 vertex $x \in S(\tilde{H})$, $k(x) \in \{0, \ldots, 4\}$. In our last step, we will perform *local routing*, in which, for all $1 \leq i \leq q$, we connect every vertex of Y_i to some vertex of $S(H_i)$ by a path, such that every vertex $x \in S(H_i)$ is an endpoint of exactly $k(x)$ such paths. We perform a similar routing in graphs of $\mathcal{H}^B$. This local routing explores the fact that every graph $\tilde{H} \in \mathcal{H}'$ is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected, together with the embedding $\mathcal{P}$ of the graph $H' = \bigcup_{\tilde{H} \in \mathcal{H}'} H'$ into H that we have computed.

In order to simplify the notation, for all $1 \leq i \leq q$, we denote by $\hat{E}_i \subseteq \hat{E}$ the set of all meta-edges of $\hat{G}$ that are incident to supernode v_i in $\hat{G}$. Similarly, for all $1 \leq i' \leq q'$, we denote by $\hat{E}'_{i'} \subseteq \hat{E}$ the set of all meta-edges of $\hat{G}$ that are incident to supernode $v'_{i'}$ in $\hat{G}$. We prove the following lemma, that allows us to perform global routing.

Lemma 6.9 *For every meta-edge $e = (v_i, v'_{i'}) \in \hat{E}$, there is a multiset $M'_{i,i'}$ of pairs of vertices of $S(H_i) \times S(H'_{i'})$, for which the following hold. For all $1 \leq i \leq q$, for every vertex $x \in S(H_i)$, let $k(x)$ be the total number of pairs in $\bigcup_{(v_i, v'_{i'}) \in \hat{E}_i} M'_{i,i'}$, in which vertex x participates. Similarly, for all $1 \leq i' \leq q'$, for every vertex $x \in S(H'_{i'})$, let $k(x)$ be the total number of pairs in $\bigcup_{(v_i, v'_{i'}) \in \hat{E}'_{i'}} M'_{i,i'}$, in which vertex x participates. Then:*

- *for every meta-edge $e = (v_i, v'_{i'}) \in E(\hat{G})$, a pair (x, y) of vertices may only belong to $M'_{i,i'}$ if $(x, y) \in M_{i,i'}$ (but it may be added to $M'_{i,i'}$ multiple times);*
- *for every vertex $x \in (\bigcup_{i=1}^{q} S(H_i)) \cup \left(\bigcup_{i'=1}^{q'} S(H'_{i'})\right)$, $k(x) \in \{0, \ldots, 4\}$;*
- *for all $1 \leq i \leq q$, $\sum_{x \in S(H_i)} k(x) = D_i$; and*
- *for all $1 \leq i' \leq q'$, $\sum_{x \in S(H'_{i'})} k(x) = D'_{i'}$.*

Proof: We construct the following directed flow network. We start with a bipartite graph $\tilde{G} = (\tilde{X}, \tilde{Y}, \tilde{E})$, where $\tilde{X} = \bigcup_{i=1}^{q} S(H_i)$, $\tilde{Y} = \bigcup_{i'=1}^{q'} S(H'_{i'})$, and $\tilde{E} = \bigcup_{(v_i, v'_{i'}) \in E(\hat{G})} M_{i,i'}$. All edges are directed from vertices of $\tilde{X}$ towards vertices of $\tilde{Y}$, and they have capacity 4 each. For all $1 \leq i \leq q$, we add a vertex s_i, that connects to every vertex in $S(H_i)$ with an edge of capacity 4. For all $1 \leq i' \leq q'$, we add a vertex $t_{i'}$, to which every vertex of $S(H'_{i'})$ connects with an edge of capacity 4. Lastly, we add a source vertex s, and a destination vertex t. For all $1 \leq i \leq q$, we add an edge (s, s_i) of capacity D_i, and for all $1 \leq i' \leq q'$, we add an edge (t_i, t) of capacity $D'_{i'}$.

We claim that this network as a valid s-t flow f of value $D = \sum_{i=1}^{q} D_i = \sum_{i'=1}^{q'} D'_{i'}$. We obtain this flow as follows. Consider a meta-edge $e = (v_i, v'_{i'}) \in E(\hat{G})$. Recall that we are given an integral weight $w(e) \leq N^{j-1}$, and a matching $M_{i,i'} \subseteq S(H_i) \times S(H'_{i'})$ of cardinality $\left\lceil N^{j-1}/4 \right\rceil$. For every edge $e' = (x, y) \in M_{i,i'}$, we set the flow $f(e') = \frac{w(e)}{|M_{i,i'}|} = \frac{w(e)}{\lceil N^{j-1}/4 \rceil}$. Notice that this ensures that the total flow on all edges of $M_{i,i'}$ is precisely $w(e)$, and for every edge $e' \in M_{i,i'}$, $f(e') \leq 4$. Once we process every meta-edge of $\hat{G}$, we finalize the flow values $f(e')$ for all edges $e' \in \tilde{E}$.

Consider now some index $1 \leq i \leq q$, and some vertex $x \in S(H_i)$. We claim that the total flow on all edges of $\tilde{E}$ that are incident to x in the flow network is at most 4. Indeed, recall that $\hat{E}_i$ is the set of all meta-edges of $\hat{G}$ that are incident to supernode v_i. From the definition of a routing meta-graph, we are guaranteed that $\sum_{e \in \hat{E}_i} w(e) = D_i$. For every meta-edge $e = (v_i, v'_{i'}) \in \hat{E}_i$, if some edge of $M_{i,i'}$ is incident to x, then the flow on this edge is $\frac{w(e)}{\lceil N^{j-1}/4 \rceil}$. Therefore, the total flow on all edges of $\tilde{E}$ that are incident to x is bounded by:

$$\sum_{e \in \hat{E}_i} \frac{w(e)}{\lceil N^{j-1}/4 \rceil} = \frac{D_i}{\lceil N^{j-1}/4 \rceil} \leq 4,$$

since $D_i \leq |S(H_i)| \leq N^{j-1}$ must hold.

We set the flow on edge (s_i, x) to be equal to the total amount of flow on all edges of $\tilde{E}$ that are incident to x in the flow network, which, from the above discussion, is bounded by 4.

From similar arguments, for every index $1 \leq i' \leq q'$, and every vertex $y \in S(H'_{i'})$, the total flow on all edges of $\tilde{E}$ that are incident to y in the flow network is at most 4. We set the flow on the edge (y, t_i) to be the total flow on all edges of $\tilde{E}$ that are incident to y in the flow network.

Next, we consider an index $1 \leq i \leq q$. We set the flow on edge (s, s_i) to be D_i. We claim that $\sum_{x \in S(H_i)} f(s_i, x) = D_i$. Indeed, from our construction:

$$\sum_{x \in S(H_i)} f(s_i, x) = \sum_{(v_i, v'_{i'}) \in \hat{E}_i} \sum_{(x,y) \in M_{i,i'}} f(x, y) = \sum_{e=(v_i, v'_{i'}) \in \hat{E}_i} \sum_{(x,y) \in M_{i,i'}} \frac{w(e)}{\lceil N^{j-1}/4 \rceil} = \sum_{e=(v_i, v'_{i'}) \in \hat{E}_i} w(e) = D_i.$$

(we have used the fact that $|M_{i,i'}| = \lceil N^{j-1}/4 \rceil$ for every meta-edge $(v_i, v'_{i'}) \in E(\hat{G})$).

Similarly, we consider an index $1 \leq i' \leq q'$. We set the flow on edge $(t_{i'}, t)$ to be $D'_{i'}$. Using the same reasoning as above, $\sum_{y \in S(H'_{i'})} f(y, t_i) = D'_{i'}$. We conclude that we have obtained a valid s-t flow in the above flow network, whose value is D. Since all edge capacities in the flow network are integral, from the integrality of flow, there is an integral s-t flow f' of value D in this flow network.

We are now ready to define the multisets $M'_{i,i'}$ of pairs of vertices from $S(H_i) \times S(H'_{i'})$, for all $(v_i, v'_{i'}) \in E(\hat{G})$. Consider any meta-edge $(v_i, v'_{i'}) \in E(\hat{G})$, and some pair $(x, y) \in M_{i,i'}$ of vertices. If $f'(x, y) > 0$, then we include $f'(x, y)$ copies of the pair (x, y) to $M'_{i,i'}$.

We now verify that all requirements hold for this definition of the multisets $M'_{i,i'}$ for all $(v_i, v'_{i'}) \in E(\hat{G})$. Clearly, a pair (x, y) of vertices may only be added to $M'_{i,i'}$ if $(x, y) \in M_{i,i'}$.

Consider now some vertex $x \in \bigcup_{i=1}^{q} S(H_i)$. Recall that $k(x)$ is the total number of pairs in $\bigcup_{(v_i, v'_{i'}) \in \hat{E}} M'_{i,i'}$ in which x participates. This is equal to the total flow leaving vertex x in f', which, in turn, is equal to the flow on edge (s_i, x). From our definition, the capacity of this edge is 4, so $k(x) \in \{0, \ldots, 4\}$. If $x \in \bigcup_{i'=1}^{q'} S(H'_{i'})$, then $k(x) \in \{0, \ldots, 4\}$ for similar reasons.

Consider now some index $1 \leq i \leq q$. From the above discussion, $\sum_{x \in S(H_i)} k(x) = \sum_{x \in S(H_i)} f'(s_i, x)$. In other words, $\sum_{x \in S(H_i)} k(x)$ is the total amount of flow leaving vertex s_i in f'. From conservation of flow this must be equal to the total amount of flow entering s_i. Since we send $D = \sum_{i=1}^{q} D_i$ flow units from s to t, and since, for all $1 \leq i \leq q$, the capacity of the edge (s, s_i) is D_i, we must send D_i flow units on edge (s, s_i). In other words, for all $1 \leq i \leq q$, $\sum_{x \in S(H_i)} k(x) = D_i$ must hold. From similar arguments, for all $1 \leq i' \leq q'$, $\sum_{x \in S(H'_{i'})} k(x) = D'_{i'}$ must hold. □

We are now ready to define the global routing. For every meta-edge $e = (v_i, v'_{i'}) \in E(\hat{G})$, we consider the resulting collection $M'_{i,i'} \subseteq S(H_i) \times S(H_{i'})$ of pairs of vertices. We define a (multi)-set $\mathcal{Q}'_{i,i'}$ of paths, as follows. Consider any pair $(x, y) \in M'_{i,i'}$, and assume that the number of times that it appears in $M'_{i,i'}$ is $N(x, y)$. Recall that $(x, y) \in M_{i,i'}$ must hold, so there must be a path $Q(x, y) \in \mathcal{Q}_{i,i'}$ connecting x to y. We add $N(x, y)$ copies of this path to $\mathcal{Q}'_{i,i'}$. Note that, from Lemma 6.9, $N(x, y) \leq k(x) \leq 4$ must hold.

We then let $\mathcal{Q}^0 = \bigcup_{(v_i,v'_{i'}) \in E(\hat{G})} \mathcal{Q}'_{i,i'}$ (again, set $\mathcal{Q}^0$ is a multiset, so if some path appears several times in some set $\mathcal{Q}'_{i,i'}$, then it will appear several times in $\mathcal{Q}^0$.

Recall that, for every pair $H_i, H'_{i'} \in \mathcal{H}'$ of graphs, the paths in $\mathcal{Q}_{i,i'}$ have length at most d' each, and they cause congestion at most N^4 in H. Since $\mathcal{Q}'_{i,i'}$ contains at most four copies of each path in $\mathcal{Q}_{i,i'}$, and since $|E(\hat{G})| \leq |\mathcal{H}'|^2 \leq N^2$, the paths in $\mathcal{Q}^0$ cause congestion at most $4N^6$ in H, and every path has length at most d' as before. For every vertex $x \in (\bigcup_{i=1}^{q} S(H_i)) \cup \left(\bigcup_{i'=1}^{q'} S(H'_{i'})\right)$, we use the definition of the value $k(x)$ from Lemma 6.9. The number of paths in $\mathcal{Q}^0$, in which x serves as an endpoint is then precisely $k(x)$, and, from Lemma 6.9, $k(x) \in \{0, \ldots, 4\}$. Recall also that, for all $1 \leq i \leq q$, $\sum_{x \in S(H_i)} k(x) = D_i$, and for all $1 \leq i' \leq q'$, $\sum_{x \in S(H'_{i'})} k(x) = D'_{i'}$.

Step 3: Local Routing

Consider some graph $H_i \in \mathcal{H}^A$. We have defined a set $Y_i \subseteq S(H_i)$ of D_i vertices of H_i that need to be routed. For every vertex $x \in S(H_i)$, we are also now given a value $k(x) \in \{0, \ldots, 4\}$, which is exactly the number of paths in $\mathcal{Q}^0$ for which vertex x serves as an endpoint. We are also guaranteed that $\sum_{x \in S(H_i)} k(x) = D_i$. We can then construct four sets $Z_i^1, Z_i^2, Z_i^3, Z_i^4$ of vertices of $S(X)$, such that for every vertex $x \in S(X)$, each of the four sets contains at most one copy of x; and the number of sets in $\left\{Z_i^1, \ldots, Z_i^4\right\}$ containing x is exactly $k(x)$. Clearly, $\sum_{a=1}^{4} |Z_i^a| = D_i$. We also partition the set Y_i of vertices into four subsets $Y_i^1, \ldots, Y_i^4$ arbitrarily, so that, for all $1 \leq a \leq 4$, $|Y_i^a| = |Z_i^a|$. Since $Y_i \subseteq S(H_i)$, from the fact that graph H_i is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected with respect to $S(H_i)$, for all $1 \leq a \leq 4$, there is a set $\hat{\mathcal{P}}_i^a$ of $|Z_i^a|$ paths in graph H_i, routing every vertex of Y_i^a to a distinct vertex of Z_i^a, such that the length of every path is at most $\tilde{d}_{j-1}$, and the paths cause congestion at most η_{j-1} in graph H_i. Let $\hat{\mathcal{P}}_i = \bigcup_{a=1}^{4} \hat{\mathcal{P}}_i^a$. We think of the paths in $\hat{\mathcal{P}}_i$ as being directed away from vertices of Y_i. Notice that the paths in $\hat{\mathcal{P}}_i$ route every vertex of Y_i to some vertex of $S(H_i)$, such that, for every vertex $x \in S(H_i)$, exactly $k(x)$ paths of $\hat{\mathcal{P}}_i$ terminate at x. The paths in $\hat{\mathcal{P}}_i$ cause congestion at most $4\eta_{j-1}$ in graph H_i, and have length at most $\tilde{d}_{j-1}$ each.

Consider the graph $H' = \bigcup_{H_i \in \mathcal{H}'} H_i$. Recall that we have computed, in Phase 1 of the algorithm, an embedding $\mathcal{P}$ of H' into H, where every path in the embedding has length at most d', and the paths in $\mathcal{P}$ cause congestion at most $N^{128\epsilon^2}$ in H.

Consider now the set $\hat{\mathcal{P}} = \bigcup_{1 \leq i \leq q} \hat{\mathcal{P}}_i$ of paths in graph H'. This set of paths routes every vertex of $\bigcup_{1 \leq i \leq q} Y_i$ to some vertex of $\bigcup_{1 \leq i \leq q} S(H_i)$, such that, for every vertex $x \in \bigcup_{1 \leq i \leq q} S(H_i)$, exactly $k(x)$ paths of $\hat{\mathcal{P}}$ terminate at x. Additionally, the length of every path in $\hat{\mathcal{P}}$ is is at most $\tilde{d}_{j-1}$, and the paths in $\hat{\mathcal{P}}$ cause congestion at most $4\eta_{j-1}$ in H'.

We now use the algorithm from Observation 3.1 with the collection $\hat{\mathcal{P}}$ of paths, and the embedding $\mathcal{P}$ of graph $H' = \bigcup_{H_i \in \mathcal{H}'} H_i$ into H, in order to compute a set $\mathcal{Q}^1$ of paths in graph H, routing every vertex $\bigcup_{1 \leq i \leq q} Y_i$ to some vertex of $\bigcup_{1 \leq i \leq q} S(H_i)$, such that, for every vertex $x \in \bigcup_{1 \leq i \leq q} S(H_i)$, exactly $k(x)$ paths of $\hat{\mathcal{P}}$ terminate at x. The algorithm ensures that the length of every path in $\hat{\mathcal{P}}$ is at most $\tilde{d}_{j-1} \cdot d'$, and the paths in $\hat{\mathcal{P}}$ cause congestion at most $4\eta_{j-1} \cdot N^{128\epsilon^2}$.

For every index $1 \leq i' \leq q'$, we similarly compute a set $\hat{\mathcal{P}}'_{i'}$ of paths in graph $H'_{i'}$, that route some

vertices of $S(H_{i'})$ to vertices of $Y'_{i'}$, so that for every vertex $y \in Y'_{i'}$, exactly one path in $\hat{\mathcal{P}}'_{i'}$ terminates at y, and for every vertex $x \in S(H'_{i'})$, exactly $k(x)$ paths of $\mathcal{P}'_{i'}$ originate at x. We use the embedding $\mathcal{P}$ of graph H' into H exactly as before, in order to compute a set $\mathcal{Q}^2$ of paths in graph H, routing vertices of $\bigcup_{1\leq i'\leq q'} S(H'_{i'})$ to vertices of $\bigcup_{1\leq i'\leq q'} Y'_{i'}$, such that, for every vertex $x \in \bigcup_{1\leq i\leq q} S(H_i)$, exactly $k(x)$ paths of $\mathcal{Q}^2$ originate at x, and for every vertex $y \in \bigcup_{1\leq i'\leq q'} Y'_{i'}$, exactly one path of $\mathcal{Q}^2$ terminates at y. As before, we can ensure that the length of each every path in $\mathcal{Q}^2$ is at most $\tilde{d}_{j-1}\cdot d'$, and the paths in $\mathcal{Q}^2$ cause congestion at most $4\eta_{j-1}\cdot N^{128\epsilon^2}$ in H.

By concatenating the paths of $\mathcal{Q}^1, \mathcal{Q}^0$ and $\mathcal{Q}^2$, we obtain the final set $\mathcal{P}''$ of paths, that defines a one-to-one routing between vertices of $\bigcup_{i=1}^{q} Y_i$ and vertices of $\bigcup_{i'=1}^{q'} Y'_{i'}$. The length of every path in $\mathcal{P}''$ is bounded by $2\tilde{d}_{j-1}\cdot d' + d' \leq 3\tilde{d}_{j-1}\cdot d'$.

Recall that $\tilde{d}_{j-1} = 2^{c(j-1)/\epsilon^4}$, while $d' < \frac{1}{4}\cdot 2^{c/\epsilon^4}$ from Inequality 5. Therefore, we get that the length of every path in $\mathcal{P}''$ is at most:

$$3\tilde{d}_{j-1}\cdot d' \leq 2^{c(j-1)/\epsilon^3}\cdot 2^{c/\epsilon^3} = 2^{cj/\epsilon^3} = \tilde{d}_j.$$

The total congestion caused by the paths in $\mathcal{P}''$ is bounded by:

$$4N^6 + 8\eta_{j-1}\cdot N^{128\epsilon^2} = 4N^6 + 8\cdot N^{6+256(j-1)\epsilon^2}\cdot N^{128\epsilon^2} \leq \frac{N^{6+256j\epsilon^2}}{2} = \frac{\eta_j}{2},$$

since $\eta_{j-1} = N^{6+256(j-1)\epsilon^2}$, and N is sufficiently large.

This completes the proof of Lemma 6.7

7 APSP in Well-Connected Graphs – Proof of Theorem 2.3

The goal of this section is to prove Theorem 2.3. We do so using the following theorem.

Theorem 7.1 *There are large enough constants c', c'', and a deterministic algorithm, whose input consists of:*

- *a parameter $0 < \epsilon < 1/400$;*
- *a pair N, $1 \leq j \leq \lceil 1/\epsilon \rceil$ of integers, such that N is sufficiently large, so that $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^6}$ holds;*
- *a graph H with $|V(H)| = N^j$; and*
- *a level-j Hierarchical Support Structure for graph H, such that graph H is $(\eta_j, \tilde{d}_j)$-well-connected with respect to the set $S(H)$ of vertices defined by the Hierarchical Support Structure.*

Further, we assume that graph H undergoes an online sequence of less than $\Lambda_j = N^{j-8-300j\epsilon^2}$ edge deletions. The algorithm maintains a set $S'(H) \subseteq S(H)$ of vertices of H, called supported vertices, *such that, at the beginning of the algorithm, $S'(H) = S(H)$, and over the course of the algorithm, vertices can leave $S'(H)$ but they may not join it. The algorithm ensures that $|S'(H)| \geq \frac{N^j}{16^j}$ holds over the course of the algorithm, and it supports short-path queries between supported vertices: given a pair $x, y \in S'(H)$ of vertices, return a path P connecting x to y in the current graph H, whose*

length is at most $d^*_j = 2^{c'j/\epsilon^5}$, in time $O(|E(P)|)$. The total update time of the algorithm is bounded by $2c''jN^{j+3} \cdot 2^{4c'/\epsilon^6} + c''m \cdot N^2 \cdot 2^{4c'/\epsilon^6}$, where m is the number of edges in graph H at the beginning of the algorithm. (If $\Lambda_j \leq 1$, then the algorithm only needs to support short-path queries until the first edge deletion).

It is immediate to verify that Theorem 2.3 follows from Theorem 7.1, by substituting $j = 1/\epsilon$. In the remainder of this section we prove Theorem 7.1. The proof is by induction on j.

7.1 Base Case: $j \leq 8$

We first consider the base case, where $j \leq 8$. In this case, $\Lambda_j \leq 1$ holds, and so we only need to support short-path queries until the first edge deletion.

In this case, the level-j Hierarchical Support Structure for graph H defines a set $S(H)$ of vertices, and, from Claim 5.1, $|V(H) \setminus S(H)| \leq |V(H)| \cdot \frac{4j}{N^{\epsilon^4}}$. Since $j \leq \lceil 1/\epsilon \rceil$, and $N^{\epsilon^4} \geq 2^{128/\epsilon^5}$ from the statement of Theorem 7.1, we get that $|S(H)| \geq |V(H)|/2 = N^j/2$. We set $S'(H) = S(H)$, and this set remains unchanged throughout the algorithm.

Recall that are guaranteed that graph H is $(\eta_j, \tilde{d}_j)$-well-connected with respect to $S(H)$, where $\tilde{d}_j = 2^{cj/\epsilon^4} \leq \frac{2^{c'j/\epsilon^5}}{2} = \frac{d^*_j}{2}$ (if $c' > c$), and $\eta_j = N^{6+256j\epsilon^2}$. In particular, for every pair $x, y \in S'(H)$ of supported vertices, there is a path of length at most $d^*_j/2$ connecting x to y in H. We let $s \in S(H)$ be an arbitrary vertex, and we construct a BFS tree τ rooted at vertex s. From the above discussion, the depth of the tree is bounded by $d^*_j/2$. Computing the tree takes time $O(|E(H)|)$. In order to respond to a short-path query between a pair x, y of vertices of H, we simply compute the unique simple path P connecting x to y in the tree τ, which can be done in time $O(|E(P)|)$. Since the depth of the tree is bounded by $d^*_j/2$, the length of the path is at most d^*_j.

7.2 Step: $j > 8$

We assume that we are given a graph H with $|V(H)| = N^j$, together level-j hierarchical support structure for graph H, whose associated collection of graphs is $\mathcal{H} = \{H_1, \ldots, H_r\}$. Recall that $r = N - \left\lceil 2N^{1-\epsilon^4} \right\rceil$, and we are guaranteed that graph H is $(\eta_j, \tilde{d}_j)$-well-connected with respect to the set $S(H) = \bigcup_{H_i \in \mathcal{H}} S(H_i)$ of vertices. Additionally, the hierarchical support structure contains an embedding $\mathcal{P}$ of the graph $H' = \bigcup_{i=1}^{r} H_i$ into graph H via paths of length at most $2^{64/\epsilon^4}$, that causes congestion at most $N^{128\epsilon^2}$. For every edge $e \in E(H')$, we denote by $P(e) \in \mathcal{P}$ the unique path that serves as the embedding of e in G.

Our algorithm will maintain a set $S'(H) \subseteq S(H)$ of supported vertices, where initially $S'(H) = S(H)$. While vertices may leave set $S'(H)$ over the course of the algorithm, set $S(H)$ remains unchanged.

Our algorithm recursively applies the algorithm from Theorem 7.1 to each of the graphs in $\mathcal{H}$. When an edge $e \in E(H)$ is deleted, then for all $1 \leq i \leq r$, for every edge $e' \in E(H_i)$ whose corresponding embedding path $P(e')$ contains e, we delete edge e' from graph H_i. Since the paths in $\mathcal{P}$ cause congestion at most $N^{128\epsilon^2}$, the deletion of a single edge from graph H may trigger the deletion of up to $N^{128\epsilon^2}$ edges from graphs $H_1, \ldots, H_r$ overall. As the result of these edge deletions, the supported sets of vertices $S'(H_i)$ that the algorithm from Theorem 7.1 maintains recursively for each of the graphs $H_i \in \mathcal{H}$ may need to be updated. Once a graph $H_i \in \mathcal{H}$ undergoes $\lceil \Lambda_{j-1} \rceil$ edge deletions, we say that it is *destroyed*. Once graph H_i is destroyed, the corresponding set $S'(H_i)$ of vertices is set to $\emptyset$.

Our algorithm maintains a partition of the set $\mathcal{H}$ of graphs into three subsets: set $\mathcal{H}^D$ of *destroyed graphs*, set $\mathcal{H}^I$ of *inactive graphs*, and set $\mathcal{H}^A$ of active graphs. Set $\mathcal{H}^D$ contains all graphs that have been destroyed so far. Once a graph joins set $\mathcal{H}^D$, it remains in $\mathcal{H}^D$ for the remainder of the algorithm. We define the sets $\mathcal{H}^I$ and $\mathcal{H}^A$ of graphs later. We will ensure that the set $\mathcal{H}^A$ of active graphs is decremental, and we will set $S'(H) = \bigcup_{H_i \in \mathcal{H}^A} S'(H_i)$ throughout the algorithm. Intuitively, we will maintain, for every active graph $H_i \in \mathcal{H}^A$, an ES-Tree data structure that is rooted at the vertices of $S'(H_i)$. We need the following simple observation bounding the number of graphs in $\mathcal{H}^D$.

Observation 7.2 *Over the course of the algorithm, $|\mathcal{H}^D| < N/32$ always holds.*

Proof: Assume otherwise, and consider the first time t during the algorithm when $|\mathcal{H}^D| \geq N/32$ held. Recall that a graph $H_i \in \mathcal{H}$ is destroyed once it undergoes $\lceil \Lambda_{j-1} \rceil$ edge deletions. Therefore, at least $\frac{N}{32} \cdot \lceil \Lambda_{j-1} \rceil$ edges have been deleted from $\bigcup_{H_i \in \mathcal{H}} E(H_i)$ by time t. On the other hand, the deletion of a single edge from H may trigger the deletion of at most $N^{128\epsilon^2}$ edges from graphs $H_1, \ldots, H_r$. Therefore, the number of edges that have been deleted from H by time t is at least:

$$\frac{N}{32N^{128\epsilon^2}} \cdot \lceil \Lambda_{j-1} \rceil = \frac{N^{1-128\epsilon^2}}{32} \cdot \left\lceil N^{j-9-300(j-1)\epsilon^2} \right\rceil > N^{j-8-300j\epsilon^2} = \Lambda_j,$$

a contradiction. □

Consider now a graph $H_i \in \mathcal{H} \setminus \mathcal{H}^D$ at some time during the algorithm's execution. For every vertex $s \in S'(H_i)$, we denote by $\mathcal{G}(s)$ the set of all graphs $H_{i'} \in \mathcal{H} \setminus \mathcal{H}^D$, such that $S'(H_{i'}) \cap B_H(s, d^*_j/32) \neq \emptyset$. In other words, set $\mathcal{G}(s)$ contains all graphs $H_{i'}$ that have not been destroyed yet, such that some vertex in the current set $S'(H_{i'})$ of supported vertices is sufficiently close to s in the current graph H. Recall that the set $S'(H_i)$ of vertices is decremental. The following observation will be useful for us.

Observation 7.3 *Consider any time t during the algorithm's execution. Let $H_i \in \mathcal{H} \setminus \mathcal{H}^D$ be a graph that has not been destroyed by time t, and let $s \in S'(H_i)$ be any vertex in the current set of supported vertices for H_i. Then, since the beginning of the algorithm and until time t, the collection $\mathcal{G}(s)$ of graphs has been decremental: that is, graphs may have left it, but no graph may have joined it since the beginning of the algorithm.*

Proof: Assume for contradiction that $H_{i'} \in \mathcal{H}$ is some graph that did not belong to set $\mathcal{G}(s)$ at time t', but belongs to set $\mathcal{G}(s)$ at time t'', where $t' < t'' \leq t$.

From the definition, at time t'', $H_{i'} \in \mathcal{H} \setminus \mathcal{H}^D$ held. Since graphs may leave set $\mathcal{H} \setminus \mathcal{H}^D$ (when they are destroyed) but they may never join $\mathcal{H} \setminus \mathcal{H}^D$ over the course of the algorithm, we get that at time t', $H_{i'} \in \mathcal{H} \setminus \mathcal{H}^D$ held. Furthermore, from the definition, at time t'', some vertex $x \in S'(H_{i'})$ belonged to $B_H(s, d^*_j/32)$. Since set $S'(H_{i'})$ of vertices is decremental, $x \in S'(H_{i'})$ held at time t'. Since distances in graph H may only grow over time, $x \in B_H(s, d^*_j/32)$ held at time t'. Therefore, $S'(H_{i'}) \cap B_H(s, d^*_j/32) \neq \emptyset$ must have held at time t', and graph $H_{i'}$ must have belonged to $\mathcal{G}(s)$ at time t', a contradiction. □

Throughout the algorithm, the set $\mathcal{H}^I$ of inactive graphs will only contain graphs $H_i \in \mathcal{H}$ that have not been destroyed yet, for which the following property holds:

P1. For every vertex $s \in S'(H_i)$, $|\mathcal{G}(s)| \leq 7N/8$.

We note that it is possible that some graph $H_i \in \mathcal{H} \setminus \mathcal{H}^D$ has Property P1 but is not added to $\mathcal{H}^I$.

The following observation shows that once property $P1$ holds for some graph H_i, it will continue to hold until the algorithm terminates or the graph is destroyed.

Observation 7.4 *Let $H_i \in \mathcal{H}$ be any graph, and assume that Property P1 holds for H_i at some time t during the algorithm's execution. Then Property P1 holds for H_i until the algorithm terminates or until H_i is destroyed.*

Proof: Assume that Property P1 holds for graph $H_i \in \mathcal{H}$ at some time t during the algorithm's execution, and consider some time $t' > t$ during the algorithm's execution. We assume that graph H_i is not destroyed at time t', and prove that Property P1 continues to hold for H_i at time t'. Indeed, consider any vertex s that lies in set $S'(H_i)$ at time t'. Since set $S'(H_i)$ is decremental, vertex s lied in $S'(H_i)$ at time t. Since Property P1 held for graph H_i at time t, $|\mathcal{G}(s)| \leq 7N/8$ held at time t. From Observation 7.3, set $\mathcal{G}(s)$ is decremental, so $|\mathcal{G}(s)|$ may not grow between time t and time t'. Therefore, $|\mathcal{G}(s)| \leq 7N/8$ holds at time t'. We conclude that Property P1 continues to hold for H_i at time t'. □

To summarize, over the course of the algorithm, we maintain a partition of the collection $\mathcal{H}$ of graphs into three subsets: the set $\mathcal{H}^D$ of destroyed graphs; the set $\mathcal{H}^I$ of inactive graphs; and the set $\mathcal{H}^A$ of all remaining graphs, that are called *active graphs*. We ensure that every graph in $\mathcal{H}^I$ has Property P1. We also ensure that the set $\mathcal{H}^A$ of active graphs is decremental – graphs may leave it but they may not join it over time. The following claim bounds the cardinality of the collection $\mathcal{H}^I$ of graphs.

Claim 7.5 *Over the course of the algorithm, $|\mathcal{H}^I| \leq N/32$ must hold.*

Proof: Assume otherwise, and consider the first time t during the algorithm when $|\mathcal{H}^I| > N/32$ held. We will construct two large sets T_1, T_2 of vertices, so that the distance between the vertices of T_1 and the vertices of T_2 is large in the current graph H. We will then reach a contradiction by using the facts that, at the beginning of the algorithm, graph H was well-connected with respect to $S(H)$, and that the number of edges that were deleted from H is relatively small.

Recall that $|\mathcal{H}| = r = N - \left\lceil 2N^{1-\epsilon^4} \right\rceil \geq 63N/64$, since $N^{\epsilon^4} \geq 2^{128/\epsilon^5}$ from the statement of Theorem 7.1. Additionally, from Observation 7.2, $|\mathcal{H}^D| < N/32$ holds at time t. Therefore, at time t, $|\mathcal{H} \setminus \mathcal{H}^D| \geq \frac{63N}{64} - \frac{N}{32} \geq \frac{61N}{64}$. Let $\tilde{\mathcal{H}} \subseteq \mathcal{H} \setminus \mathcal{H}^D$ be a collection of graphs that is obtained as follows. We start with $\tilde{\mathcal{H}} = \mathcal{H} \setminus \mathcal{H}^D$. We then consider the graphs of $\tilde{\mathcal{H}}$ one by one, starting with the graphs of $\mathcal{H}^A$. If, when graph H_i is considered, $|\tilde{\mathcal{H}}| > \left\lceil \frac{61N}{64} \right\rceil$ holds, then we discard H_i from set $\tilde{\mathcal{H}}$. Otherwise, we terminate the algorithm with the final collection $\tilde{\mathcal{H}}$ of graphs, whose cardinality must be $\left\lceil \frac{61N}{64} \right\rceil$. Notice that, if any graph of $\mathcal{H}^A$ lies in $\tilde{\mathcal{H}}$, then $\mathcal{H}^I \subseteq \tilde{\mathcal{H}}$. Recall that, from the induction hypothesis, for every graph $H_i \in \mathcal{H} \setminus \mathcal{H}^D$, $|S'(H_i)| \geq \frac{N^{j-1}}{16^{j-1}}$ holds throughout the algorithm. We construct a set T of vertices as follows. For every graph $H_i \in \tilde{\mathcal{H}}$, we let $S''(H_i)$ be an arbitrary collection of $\left\lceil \frac{N^{j-1}}{16^{j-1}} \right\rceil$ vertices that lie in set $S'(H_i)$ at time t. We then let $T = \bigcup_{H_i \in \tilde{\mathcal{H}}} S''(H_i)$. Clearly:

$$|T| = \left\lceil \frac{61N}{64} \right\rceil \cdot \left\lceil \frac{N^{j-1}}{16^{j-1}} \right\rceil.$$

Intuitively, we would like to apply Procedure ProcSeparate from Lemma 3.10 to graph H, set T of terminals, and distance parameters $\Delta = 64/\epsilon^2$ and $d = \frac{d_j^*}{64\Delta}$, in order to compute two large subsets $T_1, T_2 \subseteq T$ of vertices with $\mathsf{dist}_H(T_1, T_2) \geq d$. However, the procedure may instead return a single vertex $s \in T$, for which the ball $B_H(s, d_j^*/64)$ contains many vertices of T. In the next observation we show that this is impossible, that is, for every vertex $s \in T$, $|B_H(s, d_j^*/64) \cap T|$ is sufficiently small.

Observation 7.6 *At time t, for every vertex $s \in T$, $|B_H(s, d_j^*/64) \cap T| < |T| \cdot \left(1 - \frac{1}{256}\right)$ holds.*

Proof: Consider some vertex $s \in T$, and denote $B = B_H(s, d_j^*/64)$. Let $T' = \bigcup_{H_i \in \mathcal{H}^I \cap \tilde{\mathcal{H}}} S''(H_i)$. Assume first that B does not contain any vertex of T'. Notice that in this case, at least one graph of $\mathcal{H}^A$ lies in $\tilde{\mathcal{H}}$, and so $\mathcal{H}^I \subseteq \tilde{\mathcal{H}}$. Since we have assumed that $|\mathcal{H}^I| > \frac{N}{32}$, we get that:

$$|\tilde{\mathcal{H}} \setminus \mathcal{H}^I| \leq |\tilde{\mathcal{H}}| - |\mathcal{H}^I| \leq \left\lceil \frac{61N}{64} \right\rceil - \frac{N}{32} \leq \left\lceil \frac{61N}{64} \right\rceil \cdot \left(1 - \frac{1}{61}\right) \leq \left\lceil \frac{61N}{64} \right\rceil \cdot \left(1 - \frac{1}{256}\right).$$

Therefore:

$$|B \cap T| \leq |T| - |T'| \leq |\tilde{\mathcal{H}} \setminus \mathcal{H}^I| \cdot \left\lceil \frac{N^{j-1}}{16^{j-1}} \right\rceil \leq \left\lceil \frac{61N}{64} \right\rceil \cdot \left(1 - \frac{1}{256}\right) \cdot \left\lceil \frac{N^{j-1}}{16^{j-1}} \right\rceil \leq |T| \cdot \left(1 - \frac{1}{256}\right).$$

Assume now that B contains at least one vertex of T', and let s' be any such vertex. Note that $B = B_H(s, d_j^*/64) \subseteq B_H(s', d_j^*/32)$. Observe that s' must lie in the current set $S'(H_i)$ of supported vertices of some graph H_i that currently lies in $\mathcal{H}^I$. From Property P1, $|\mathcal{G}(s')| \leq \frac{7N}{8}$. Let $\mathcal{G}'(s') = \mathcal{H} \setminus \mathcal{G}(s')$, so $|\mathcal{G}'(s')| \geq \frac{N}{8}$.

From our definitions, for every graph $H_{i'} \in \mathcal{G}'(s')$, $S'(H_{i'}) \cap B_H(s', d_j^*/32) = \emptyset$, and therefore, $S'(H_{i'}) \cap B = \emptyset$.

Let $\tilde{\mathcal{H}}' = \tilde{\mathcal{H}} \cap \mathcal{G}'(s')$. On the one hand, since $|\tilde{\mathcal{H}}| = \lceil \frac{61N}{64} \rceil$, and $|\mathcal{G}'(s')| \geq \frac{N}{8}$, while $|\mathcal{H}| = N$, we get that $|\tilde{\mathcal{H}}'| \geq \frac{5N}{64} \geq \lceil \frac{61N}{64} \rceil \cdot \frac{1}{256}$. On the other hand, the vertices of $\bigcup_{H_i \in \tilde{\mathcal{H}}'} S''(H_i)$ may not lie in set B. Therefore:

$$|B \cap T| \leq \left(|\tilde{\mathcal{H}}| - |\tilde{\mathcal{H}}'|\right) \cdot \left\lceil \frac{N^{j-1}}{16^{j-1}} \right\rceil \leq \left\lceil \frac{61N}{64} \right\rceil \cdot \left(1 - \frac{1}{256}\right) \cdot \left\lceil \frac{N^{j-1}}{16^{j-1}} \right\rceil \leq |T| \cdot \left(1 - \frac{1}{256}\right).$$

□

We apply Algorithm ProcSeparate from Lemma 3.10 to graph H, set T of terminals, distance parameters $\Delta = 64/\epsilon^2$, and $d = \frac{d_j^*}{64\Delta}$, and parameter $\alpha = \left(1 - \frac{1}{256}\right)$. From Observation 7.6, the algorithm may not return a vertex $s \in T$ with $|B_H(s, \Delta \cdot d)| = |B_H(s, d_j^*/64)| > \alpha \cdot |T|$. Therefore, it must compute two subsets T_1, T_2 of vertices with $|T_1| = |T_2|$, such that $|T_1| \geq \frac{|T|^{1-64/\Delta}}{256}$, and for every pair $s \in T_1, s' \in T_2$ of terminals, $\mathsf{dist}_H(s, s') \geq d$. We will now exploit the facts that graph H was well-connected with respect to set $S(H)$ of vertices at the beginning of the algorithm, and that relatively few edges were deleted from H, in order to reach a contradiction.

Observe first that:

$$|T_1| \geq \frac{|T|^{1-\epsilon^2}}{256} \geq \frac{1}{256} \cdot \left(\frac{61 \cdot N^j}{4 \cdot 16^j}\right)^{1-\epsilon^2} \geq \frac{N^{j(1-\epsilon^2)}}{1024 \cdot 16^j}.$$

On the other hand:

$$d = \frac{d_j^*}{64\Delta} = \frac{2^{c'j/\epsilon^5} \cdot \epsilon^2}{2^{12}} > 2^{cj/\epsilon^4} = \tilde{d}_j.$$

Let $H^{(0)}$ denote the graph H at the beginning of the algorithm, and let $H^{(t)}$ denote graph H at time t. Recall that, at the beginning of the algorithm, graph $H^{(0)}$ was $(\eta_j, \tilde{d}_j)$-well-connected with respect to the set $S(H) = \bigcup_{H_i \in \mathcal{H}} S(H_i)$ of vertices. Since the sets $S'(H_i)$ of vertices for graphs $H_i \in \mathcal{H}$ are

decremental, and since $S'(H_i) = S(H_i)$ at the beginning of the algorithm for each such graph, we get that $T \subseteq S(H)$ held at the beginning of the algorithm. Therefore, there was a collection $\mathcal{P}(T_1, T_2)$ of paths in graph $H^{(0)}$, routing every vertex of T_1 to a distinct vertex of T_2, such that the paths in $\mathcal{P}(T_1, T_2)$ cause congestion at most $\eta_j = N^{6+256j\epsilon^2}$, and the length of every path is at most $\tilde{d}_j$.

Let E' be the set of edges that have been deleted from graph H by time t. Note that, in graph $H^{(t)}$, no path of length at most $\tilde{d}_j$ connecting a vertex of T_1 to a vertex of T_2 exists. Therefore, set E' must contain at least one edge from every path in $\mathcal{P}(T_1, T_2)$. We conclude that:

$$|E'| \geq \frac{|T_1|}{\eta_j} \geq \frac{N^{j(1-\epsilon^2)}}{1024 \cdot 16^j \cdot N^{6+256j\epsilon^2}} = \frac{N^{j-257j\epsilon^2-6}}{1024 \cdot 16^j} > N^{j-8-257j\epsilon^2} \geq \Lambda_j,$$

since $j \leq \lceil 1/\epsilon \rceil$, $\epsilon \leq 1/400$, and $N^{\epsilon} > 2^{8/\epsilon}$ from the statement of Theorem 7.1. This is a contradiction, since fewer than Λ_j edges may be deleted from H. □

From Observation 7.2 and Claim 7.5, throughout the algorithm, $|\mathcal{H}^I| + |\mathcal{H}^D| \leq N/16$ holds. Since $|\mathcal{H}| = r = N - \left\lceil 2N^{1-\epsilon^4} \right\rceil \geq 63N/64$, we get that, throughout the algorithm:

$$|\mathcal{H}^A| \geq |\mathcal{H}| - |\mathcal{H}^I| - |\mathcal{H}^D| \geq \frac{63N}{64} - \frac{N}{16} \geq \frac{59N}{64}. \tag{10}$$

The set $S'(H)$ that we maintain throughout the algorithm is defined to be: $S'(H) = \bigcup_{H_i \in \mathcal{H}^A} S'(H_i)$. Since, for every graph H_i, set $S'(H_i)$ of vertices is decremental, and since the collection $\mathcal{H}^A$ of graphs is decremental, we get that the set $S'(H)$ of vertices is decremental as well. It is also easy to see that $S'(H) = S(H) = \bigcup_{H_i \in \mathcal{H}} S(H)$ hods at the beginning of the algorithm. Since, from Inequality 10, $|\mathcal{H}^A| \geq \frac{59N}{64}$ holds throughout the algorithm, and since, from the induction hypothesis, for every graph $H_i \in \mathcal{H}^A$, $|S'(H_i)| \geq \frac{N^{j-1}}{16^{j-1}}$ holds, we get that, throughout the algorithm:

$$|S'(H)| \geq |\mathcal{H}^A| \cdot \frac{N^{j-1}}{16^{j-1}} \geq \frac{59N}{64} \cdot \frac{N^{j-1}}{16^{j-1}} \geq \frac{N^j}{16^j},$$

as required.

7.2.1 Data Structures and Initialization

Consider a graph $H_i \in \mathcal{H}$. Note that, as part of the level-j hierarchical support structure for H, we are given a level-$(j-1)$ hierarchical support structure for H_i, and we are guaranteed that H_i is $(\eta_{j-1}, \tilde{d}_{j-1})$-well-connected with respect to $S(H_i)$. Therefore, from the induction hypothesis, we can apply the algorithm from Theorem 7.1 with parameter $(j-1)$ to graph H_i, and the corresponding level-$(j-1)$ hierarchical support structure. Parameters N and ϵ remain unchanged. We denote the corresponding data structure by $\mathcal{D}_{j-1}(H_i)$.

As part of the initialization procedure, for every graph $H_i \in \mathcal{H}$, we initialize the corresponding data structure $\mathcal{D}_{j-1}(H_i)$.

Our algorithm also maintains, for every edge $e \in E(H)$, a list $L(e)$ of edges $e' \in \bigcup_{H_i \in \mathcal{H}} E(H_i)$, such that path $P(e')$ contains e. Together with this list, we maintain a pointer from e to each such edge e' in its corresponding graph H_i, and a pointer in the opposite direction. We initialize the lists $L(e)$ for edges $e \in E(H)$ at the beginning of the algorithm.

We also initialize $\mathcal{H}^A = \mathcal{H}$, $\mathcal{H}^I = \mathcal{H}^D = \emptyset$, and $S'(H) = S(H)$.

Lastly, for every graph $H_i \in \mathcal{H}$, we initialize an ES-Tree data structure, whose corresponding tree is denoted by τ_i, that, intuitively, is rooted at the set $S'(H_i)$ of vertices. Specifically, in order to construct data structure τ_i, we let $\tilde{G}_i$ be a graph that is obtained from H, after we add a source vertex s_i to it, which connects to every vertex in $S'(H_i)$ with an edge. We then let τ_i be an ES-Tree data structure in graph $\tilde{G}_i$, rooted at vertex s_i, with depth bound $d^*_j/8+1$. When an edge is deleted from H, we will also delete it from $\tilde{G}_i$, and update the ES-Tree data structure τ_i accordingly. When a vertex s is deleted from set $S'(H_i)$, we will delete the edge (s_i, s) from graph $\tilde{G}_i$, and update τ_i accordingly.

We maintain, for every pair of graphs $H_i \in \mathcal{H}^A$ and $H_{i'} \in \mathcal{H} \setminus \mathcal{H}^D$, a counter $n_{i,i'}$, that counts the number of vertices of $S'(H_{i'})$ that lie at distance at most $d^*_j/32+1$ from s_i in tree τ_i. Note that, if $n_{i,i'}=0$, then for every vertex $s \in S'(H_i)$, $S'(H_{i'}) \cap B_H(s, d^*_j/32) = \emptyset$. In other words, $H_{i'} \notin \mathcal{G}(s)$ holds for all $s \in S'(H_i)$. We also maintain a counter $\tilde{n}_i$, whose value is the number of graphs $H_{i'} \in \mathcal{H} \setminus \mathcal{H}^D$, with $n_{i,i'} > 0$. We can initialize the values $n_{i,i'}$ for every pair $H_i, H_{i'} \in \mathcal{H}$ of graphs, and the counters in $\{\tilde{n}_i\}_{H_i \in \mathcal{H}}$ at the beginning of the algorithm. The time that is required in order to do so is subsumed by the time required to initialize the ES-Tree data structures. We need the following simple observation.

Observation 7.7 *Let $H_i \in \mathcal{H}$ be a graph. Assume that at some point t in the algorithm's execution, $H_i \notin \mathcal{H}^D$ holds, and $\tilde{n}_i \leq 7N/8$. Then graph H_i has Property P1 at time t.*

Proof: Assume otherwise. Then at time t, there is some vertex $s \in S'(H_i)$ with $|\mathcal{G}(s)| > 7N/8$. If $H_{i'}$ is a graph that lies in $\mathcal{G}(s)$ at time t, then at least one vertex of $S'(H_{i'})$ must lie in $B_H(s, d^*_j/32)$ at time t, and so $n_{i,i'} > 0$ must hold. But then $\tilde{n}_i > 7N/8$ must hold at time t, a contradiction. □

Throughout the algorithm's execution, whenever, for some graph $H_i \in \mathcal{H}^A$, the value of counter $\tilde{n}_i$ becomes at most $7N/8$, we move graph H_i from $\mathcal{H}^A$ to $\mathcal{H}^I$. Lastly, we need the following simple observation.

Observation 7.8 *Let $H_i, H_{i'}$ be any pair of graphs that lie in set $\mathcal{H}^A$ at some time t during the algorithm's execution. Let s be any vertex that lies in $S'(H_i)$ at time t, and let s' be any vertex that lies in $S'(H_{i'})$ at time t. Then $\mathsf{dist}_H(s,s') \leq d^*_j/8$ at time t.*

Proof: Let $\mathcal{G}_i$ be the set of all graphs $H_{i''} \in \mathcal{H} \setminus \mathcal{H}^D$ with $n_{i,i''} > 0$ at time t. Since $H_i \in \mathcal{H}^A$ at time t, $|\mathcal{G}_i| \geq 7N/8$ must hold. Similarly, let $\mathcal{G}_{i'}$ be the set of all graphs $H_{i''} \in \mathcal{H} \setminus \mathcal{H}^D$ with $n_{i',i''} > 0$ at time t. As before, $|\mathcal{G}_{i'}| \geq 7N/8$. Therefore, there is some graph $H_{i''} \in \mathcal{G}_i \cap \mathcal{G}_{i'}$.

In the remainder of this proof, whenever we refer to vertex sets $S'(H_i), S'(H_{i'}), S'(H_{i''})$, or to graphs $H_i, H_{i'}, H_{i''}, H$, we mean the corresponding sets of vertices or the corresponding graphs at time t.

Since, at time t, $n_{i,i''} > 0$, there is some vertex $x \in S'(H_{i''})$, such that the distance from s_i to x in tree τ_i is at most $d^*_j/32+1$. Therefore, there is some vertex $x' \in S'(H_i)$, such that $\mathsf{dist}_H(x,x') \leq d^*_j/32$. Let Q be a path of length at most $d^*_j/32$ connecting x to x' in H. From a similar reasoning, there is a pair of vertices $y \in S'(H_{i''})$ and $y' \in S'(H_{i'})$, such that $\mathsf{dist}_H(y,y') \leq d^*_j/32$. Let Q' be a path of length at most $d^*_j/32$ connecting y to y' in H.

From the induction hypothesis, there is a path P_1 of length at most d^*_{j-1} connecting s to x' in graph H_i. Recall that we are given an embedding $\mathcal{P}$ of the edges of $\bigcup_{H_a \in \mathcal{H}} E(H_a)$ into H, where the length of every path in $\mathcal{P}$ is at most $2^{64/\epsilon^4}$. From Observation 3.1, there is a path P'_1 in graph H, connecting s to x', whose length is at most $2^{64/\epsilon^4} \cdot d^*_{j-1}$.

Using similar reasonings, there is a path P'_2 in graph H connecting x to y of length at most $2^{64/\epsilon^4} \cdot d^*_{j-1}$, and a path P'_3 in graph H connecting y' to s', of length at most $2^{64/\epsilon^4} \cdot d^*_{j-1}$. By concatenating the paths P'_1, Q, P'_2, Q', P'_3, we obtain a path in graph H, connecting s to s', whose length is bounded by:

$$\frac{d^*_j}{16} + 3 \cdot 2^{64/\epsilon^4} \cdot d^*_{j-1} = \frac{d^*_j}{16} + 3 \cdot 2^{64/\epsilon^4} \cdot 2^{c'(j-1)/\epsilon^5} \leq \frac{d^*_j}{16} + \frac{2^{c'j/\epsilon^5}}{16} \leq \frac{d^*_j}{8}$$

(since $d^*_j = 2^{c'j/\epsilon^5}$). □

Next, we describe an algorithm for updating the data structures after each edge deletion, followed by the analysis of the total update time of the algorithm. We then conclude with an algorithm for responding to queries.

7.2.2 Maintaining the Data Structures

For each graph $H_i \in \mathcal{H}$, our algorithm will only maintain ES-Tree data structure τ_i, together with the corresponding counters $\tilde{n}_i$ and $n_{i,i'}$ for all $H_{i'} \in \mathcal{H} \setminus \mathcal{H}^D$, as long as graph H_i lies in $\mathcal{H}^A$. Once graph H_i is removed from $\mathcal{H}^A$, we no longer maintain these data structures.

We now describe an algorithm for updating the data structures following the deletion of an edge e from graph H. The algorithm, called DeleteEdge(e), is straightforward and it is summarized in Figure 1.

When an edge e is deleted from graph H, we consider every edge $e' \in L(e)$ (that is, edges $e' \in \bigcup_{H_i \in \mathcal{H}} E(H_i)$, whose embedding path $P(e')$ contains edge e. We assume that $e' \in E(H_i)$, and that $H_i \in \mathcal{H} \setminus \mathcal{H}^D$ (if $H_i \in \mathcal{H}^D$, no further action is required for processing edge e'). We then delete edge e' from graph H_i and update the corresponding data structure $\mathcal{D}_{j-1}(H_i)$. As a result, we obtain a set X_i (that may be empty) of vertices that have been deleted from $S'(H_i)$. We also update lists $L(e'')$ of edges $e'' \in E(H)$ that contain e', to remove e' from these lists.

If the number of edges deleted so far from H_i reaches at least Λ_{j-1}, then graph H_i is destroyed. We add the graph to set $\mathcal{H}^D$, and we update all counters $n_{i',i}$ and $\tilde{n}_{i'}$ for graphs $H_{i'} \in \mathcal{H}^A$ as needed. Otherwise, we process every vertex $s \in X_i$ one by one. If $H_i \in \mathcal{H}^A$, then we delete edge (s_i, s) from graph $\tilde{G}_i$, and the corresponding ES-Tree data structure τ_i. As the result, some distances in tree τ_i may have increased, and we need to update all counters in $\left\{n_{i,i'}\right\}_{H_{i'} \in \mathcal{H} \setminus \mathcal{H}^D}$, as well as counter $\tilde{n}_i$ accordingly. Additionally, for every graph $H_{i'} \in \mathcal{H}^A$, if the distance from s_i to s in tree τ_i was bounded by $d^*_j/32 + 1$, then we need to decrease $n_{i',i}$ by 1 (as vertex s no longer lies in $S'(H_i)$), and if needed, we need to update counter $\tilde{n}_{i'}$.

Once we finish processing every edge $e' \in L(e)$, we also need to delete edge e from every graph $\tilde{G}_{i'}$, where $H_{i'} \in \mathcal{H}^A$, and the corresponding ES-Tree data structure $\tau_{i'}$. As before, this may increase some distances in tree $\tau_{i'}$, and we may need to update counters $\left\{n_{i',i''}\right\}_{H_{i''} \in \mathcal{H} \setminus \mathcal{H}^D}, \tilde{n}_{i'}$ accordingly. Lastly, we consider every graph $H_{i'} \in \mathcal{H}^A$ in turn. If $\tilde{n}_{i'} \leq 7N/8$ holds for any such graph, then we move it from $\mathcal{H}^A$ to $\mathcal{H}^I$.

It is easy to verify that the algoritm maintains all data structures correctly, and, from Observation 7.7, when graph H_i is added to $\mathcal{H}^I$, Property P1 holds for it. From Observation 7.4, once H_i is added to $\mathcal{H}^I$, Property P1 continues to hold for it until the end of the algorithm, or until H_i is added to $\mathcal{H}^D$. From the above discussion, we are guaranteed that, throughout the algorithm, $|\mathcal{H}^A| \neq \emptyset$.

7.2.3 Analysis of Total Update Time

Let m denote the number of edges in graph H at the beginning of the algorithm. Recall that every edge $e \in E(H)$ participates in at most $N^{128\epsilon^2}$ paths in $\mathcal{P}$. Therefore, the length of the list $L(e)$ is bounded by $N^{128\epsilon^2}$. Every edge of $\bigcup_{H_i \in \mathcal{H}} E(H_i)$ may be added at most once to list $L(e)$ when the

ALGORITHM DeleteEdge(e)

1. For every edge $e' \in L(e)$, such that the graph $H_i \in \mathcal{H}$ containing e' lies in $\mathcal{H} \setminus \mathcal{H}^D$ do:

   (a) Delete e' from H_i and update the corresponding data structure $\mathcal{D}_{j-1}(H_i)$. Let X_i be the set of vertices that were deleted from $S'(H_i)$ as the result of this update.

   (b) For every edge $e'' \in E(H)$ with $e' \in L(e'')$, delete e' from $L(e'')$.

   (c) If the number of edges deleted so far from H_i becomes at least Λ_{j-1}:

      i. Add graph H_i to $\mathcal{H}^D$ and remove it from the set $\mathcal{H}^A$ or $\mathcal{H}^I$ to which it belonged.

      ii. For every graph $H_{i'} \in \mathcal{H}^A$, if $n_{i',i} > 0$, set $n_{i',i}$ to 0 and decrease $\tilde{n}_{i'}$ by 1.

   (d) Otherwise: for every vertex $s \in X_i$ do:

      i. If $H_i \in \mathcal{H}^A$, delete edge (s_i, s) from graph $\tilde{G}_i$ and update the ES-Tree τ_i, together with counters $\left\{n_{i,i'}\right\}_{H_{i'} \in \mathcal{H} \setminus \mathcal{H}^D}, \tilde{n}_i$ accordingly.

      ii. For every graph $H_{i'} \in \mathcal{H}^A$, if $\mathsf{dist}_{\tau_{i'}}(s_{i'}, s) \leq d^*_j/32 + 1$, decrease $n_{i',i}$ by 1. If $n_{i',i}$ decreases from 1 to 0, decrease $\tilde{n}_{i'}$ by 1.

   .

2. For every graph $H_{i'} \in \mathcal{H}^A$, delete edge e from graph $\tilde{G}_{i'}$, and update the ES-Tree $\tau_{i'}$, together with counters $\left\{n_{i',i''}\right\}_{H_{i''} \in \mathcal{H} \setminus \mathcal{H}^D}, \tilde{n}_{i'}$ accordingly.

3. For every graph $H_{i'} \in \mathcal{H}^A$, if $\tilde{n}_{i'} \leq 7N/8$ holds, move $H_{i'}$ from $\mathcal{H}^A$ to $\mathcal{H}^I$.

Figure 1: Algorithm DeleteEdge(e)

data structure is initialized, and subsequently it may be deleted at most once from $L(e)$. Therefore, the total time required to maintain the lists $L(e)$ for all edges $e \in E(H)$ is at most $O(m \cdot N^{128\epsilon^2})$.

Consider now some graph $H_i \in \mathcal{H}$, and denote by m_i the number of edges in H_i at the beginning of the algorithm. Recall that, from the definition of the hierarchical support structure, $|E(H_i)| \leq N^{j-1+32\epsilon^2}$. From the induction hypothesis, maintaining data structure $\mathcal{D}_{j-1}(H_i)$ recursively takes time at most:

$$\begin{aligned} 2c''(j-1)N^{j+2} \cdot 2^{4c'/\epsilon^6} + c''|E(H_i)| \cdot N^2 \cdot 2^{4c'/\epsilon^6} &\leq 2c''(j-1)N^{j+2} \cdot 2^{4c'/\epsilon^6} + c''N^{j+1+32\epsilon^2} \cdot 2^{4c'/\epsilon^6} \\ &\leq c''(2j-1)N^{j+2} \cdot 2^{4c'/\epsilon^6}. \end{aligned}$$

Since $|\mathcal{H}| = N$, the total update time needed in order to maintain these data structures for all graphs $H_i \in \mathcal{H}$ is bounded by $c''(2j-1)N^{j+3} \cdot 2^{4c'/\epsilon^5}$.

Consider now some graph $H_i \in \mathcal{H}$. The total update time that is needed in order to maintain ES-Tree τ_i is bounded by:

$$O(jm \cdot d_j^* \cdot \log N) \leq O(jm \cdot 2^{c'j/\epsilon^5} \log N) \leq O(m \cdot 2^{4c'/\epsilon^6} \log N),$$

since $d_j^* = 2^{c'j/\epsilon^4}$ and $j \leq \lceil 1/\epsilon \rceil$. We can initialize the counters $n_{i,i'}$ for all graphs $H_{i'} \in \mathcal{H}$ and $\tilde{n}_i$ without increasing this asymptotic running time. We can also perform updates to these counters in Step 1(d)i within the same asymptotic running time. Since $|\mathcal{H}| = N$, this part of the algorithm takes total update time at most $O(N \cdot m \cdot 2^{4c'/\epsilon^6} \log N)$.

Whenever a vertex s is deleted from a set $S'(H_i)$ for any graph $H_i \in \mathcal{H}$, we may need to update the counters $n_{i',i}$ and $\tilde{n}_{i'}$ for some graphs $H_{i'} \in \mathcal{H}^A$. This can be done in time $O(N)$ per vertex. Since a vertex may be deleted at most once from $\bigcup_{H_i \in \mathcal{H}} S'(H_i)$, these updates can be done in time $O(N^{j+1})$.

Lastly, every graph $H_i \in \mathcal{H}$ may be moved to set $\mathcal{H}^D$ at most once over the course of the algorithm, at which time we may need to update counters $n_{i',i}$ and $\tilde{n}_{i'}$ for graphs $H_{i'} \in \mathcal{H}_A$. This takes time $O(N)$ per graph $H_i \in \mathcal{H}$, and $O(N^2)$ overall.

From the above discussion, the total update time of the algorithm is bounded by:

$$c''(2j-1)N^{j+3} \cdot 2^{4c'/\epsilon^6} + O(N \cdot m \cdot 2^{4c'/\epsilon^6} \log N) + O(N^{j+1}) \leq 2c''jN^{j+3} \cdot 2^{4c'/\epsilon^6} + c''m \cdot N^2 \cdot 2^{4c'/\epsilon^6}.$$

7.2.4 Response to Queries

In this subsection we describe an algorithm for responding to a short-path query between a pair $x, y \in S'(H)$ of vertices. Recall that the goal is to return a path P of length at most d_j^* connecting x to y in H, in time $O(|E(P)|)$.

Recall that $S'(H) = \bigcup_{H_i \in \mathcal{H}^A} S'(H_i)$. Therefore, there is a pair of graphs $H_i, H_{i'} \in \mathcal{H}^A$ (where possibly $H_i = H_{i'}$), with $x \in S'(H_i)$ and $y \in S'(H_{i'})$. From Observation 7.8, $\mathsf{dist}_H(x,y) \leq d_j^*/8$ must hold. In particular, if we consider the ES-Tree data structure τ_i, then $y \in V(\tau_i)$ must hold, and the distance from y to s_i in the tree must be at most $d_j^*/8 + 1$. Let Q_1 be the path connecting y to s_i in tree τ_i. We delete the last vertex on the path, and let $x' \in S'(H_i)$ be the new last vertex of Q_1. Using the induction hypothesis, we can compute a path P' in graph H_i connecting x to x', whose length is at most d_{j-1}^*. Assume that the sequence of edges on path Q' is $(e_1, e_2, \ldots, e_z)$. For all $1 \leq z' \leq z$, consider the path $P(e_{z'}) \in \mathcal{P}$ that serves as the embedding of edge z' into H. Recall that the length

of the path is bounded by $2^{64/\epsilon^4}$. By concatenating the paths $P(e_1), \ldots, P(e_z)$, we obtain a path Q_2 in graph H, connecting x to x', whose length is at most $d^*_{j-1} \cdot 2^{64/\epsilon^4}$. Lastly, by concatenating paths Q_1 and Q_2, we obtain a path P in graph H connecting x to y. The length of the path is bounded by:

$$d^*_{j-1} \cdot 2^{64/\epsilon^4} + \frac{d^*_j}{8} \leq 2^{c'(j-1)/\epsilon^5} \cdot 2^{64/\epsilon^4} + \frac{2^{c'j/\epsilon^5}}{8} \leq 2^{c'j/\epsilon^5} = d^*_j.$$

since $d^*_j = 2^{c'j/\epsilon^5}$. It is easy to see that the algorithm can be implemented in time $O(|E(P)|)$.

8 APSP in Expanders – Proof of Theorem 2.4

This section is dedicated to the proof of Theorem 2.4. We first prove the following lemma, that can be viewed as a weaker variation of Theorem 2.4, in the sense that it can only withstand a significantly shorter sequence of edge deletions.

Lemma 8.1 *There is a deterministic algorithm whose input consists of an n-vertex graph G with $|E(G)| = m$ that is a φ^*-expander, for some $0 < \varphi^* < 1$, with maximum vertex degree at most Δ, and a parameter $\frac{2}{(\log n)^{1/12}} < \epsilon < \frac{1}{400}$, such that $1/\epsilon$ is an integer. We assume that graph G undergoes an online sequence of at most $\frac{n^{1-20\epsilon}(\varphi^*)^2}{\Delta^2}$ edge deletions. The algorithm maintains a set $U \subseteq V(G)$ of vertices, such that, for every integer $t > 0$, after t edges are deleted from G, $|U| \leq \frac{4\Delta t}{\varphi^*}$ holds. Vertex set U is incremental, so vertices may join it but they may not leave it. The algorithm also supports short-path queries: given a pair of vertices $x, y \in V(G) \setminus U$, return an x-y path P in the current graph G, of length at most $\frac{2^{O(1/\epsilon^6)} \cdot \Delta \cdot \log n}{\varphi^*}$, with query time $O(|E(P)|)$. The total update time of the algorithm is $O\left(\frac{m^{1+O(\epsilon)} \cdot \Delta^3}{(\varphi^*)^2}\right)$.*

The proof of Lemma 8.1 is deferred to Section 8.1. We now complete the proof of Theorem 2.4 using it. We partition the execution of our algorithm into phases. Let $k' = \left\lfloor \frac{n^{1-20\epsilon}\varphi^2}{2^{11} \cdot \Delta^4} \right\rfloor$. The first phase lasts as long as the number of edges deleted from G via the input update sequence is at most k'. Once k' edges are deleted from graph G, the second phase begins. Each subsequent phase similarly lasts for as long as at most k' edges are deleted since the beginning of the phase, except for the last phase which may be shorter, if the input update sequence terminates before k' edges are deleted from G since the beginning of the phase. For all $i \geq 1$, we denote by Σ_i the sequence of edge deletions that graph G undergoes as part of the online input sequence of edge deletions in phase i, and we denote by E_i the set of edges that belong to Σ_i. Since the total number of edges in the input sequence of edge deletions is bounded by $\frac{n \cdot \varphi^2}{2^{13}\Delta^4}$, we get that the number of phases is bounded by $\frac{n \cdot \varphi^2}{2^{13} \cdot \Delta^4 \cdot k'} \leq \frac{n^{20\epsilon}}{4}$.

We define another dynamic graph G'. At the beginning of the algorithm, we set $G' = G$. As the algorithm progresses, we will delete some edges from G'. Specifically, at the end of every phase of the algorithm, we will define a set of edges to be deleted from graph G'; we do not delete any edges from G' as long as a phase progresses. We run the algorithm from Theorem 3.4 on graph G'. We will ensure that the number of edges that are deleted from G' over the course of the entire algorithm is bounded by $k = \frac{\varphi n}{16\Delta}$. Clearly, $k \leq \frac{\varphi \cdot |E(G)|}{10\Delta}$. From Theorem 3.4, we are guaranteed that, throughout the algorithm, $|\tilde{U}| \leq \frac{8k\Delta}{\varphi} \leq \frac{n}{2}$ holds.

In order to execute the first phase, we let $H_1 = G' = G$, and we apply the algorithm from Lemma 8.1 to graph H_1, and the online sequence Σ_1 of edge deletions. Recall that graph G is a φ-expander, and that $|\Sigma_1| \leq k' = \left\lfloor \frac{n^{1-20\epsilon}\varphi^2}{2^{11} \cdot \Delta^4} \right\rfloor$. Recall that the algorithm maintains an incremental set $U \subseteq V(G)$ of

vertices, that we denote by U_1, such that, for every integer $t > 0$, after t edges are deleted from G, $|U_1| \leq \frac{4\Delta t}{\varphi}$ holds. Throughout the first phase, we let the set U of vertices that our algorithm maintains be U_1. Recall that the algorithm from Lemma 8.1 supports queries short-path queries: given a pair of vertices $x, y \in V(G) \setminus U_1$, return an x-y path P in the current graph $H_1 = G$, of length at most $\frac{2^{O(1/\epsilon^6)} \cdot \Delta \cdot \log n}{\varphi}$, with query time $O(|E(P)|)$.

From the above discussion, at the end of the first phase, $|U_1| \leq \frac{4\Delta k'}{\varphi} \leq \frac{n^{1-20\epsilon} \cdot \varphi}{2^9 \cdot \Delta^3}$ holds. We denote by $\tilde{E}_1$ the set of all edges that are incident to the vertices of U_1 in the current graph G. We then update graph G', by deleting the edges of $E_1 \cup \tilde{E}_1$ from it. Therefore, at the end of the first phase, the number of edges that are deleted from G' is bounded by:

$$|E_1| + |\tilde{E}_1| \leq \frac{n^{1-20\epsilon} \cdot \varphi}{2^8 \cdot \Delta^3} < k.$$

We denote by $\tilde{U}_1$ the set of vertices that the algorithm from Theorem 3.4 produces at the end of Phase 1, and the deletion of the edges of $E_1 \cup \tilde{E}_1$ from G'. We also denote by $H_2 = G' \setminus \tilde{U}_1$. From Theorem 3.4, graph H_2 is a $\varphi/(6\Delta)$-expander. Notice also that, if U_1 is the set of vertices that the algorithm from Lemma 8.1 maintains at the end of the phase, then $U_1 \subseteq \tilde{U}_1$ must hold.

The remaining phases are executed similarly, with several minor differences. We let $\tilde{U}_i$ be the set $\tilde{U}$ that the algorithm from Theorem 3.4 produces at the end of Phase $(i-1)$, and we denote $H_i = G' \setminus \tilde{U}_i$. We will ensure that the total number of edges that are deleted from graph G' over the course of the first $(i-1)$ phases is bounded by:

$$(i-1) \cdot \frac{n^{1-20\epsilon} \cdot \varphi}{32 \cdot \Delta}.$$

Since the total number of phases is bounded by $\frac{n^{20\epsilon}}{4}$, this ensures that the number of edges deleted so far from G' is less than $\frac{n \cdot \varphi}{16\Delta} = k$. Therefore, from Theorem 3.4, graph H_i is a $\varphi/(6\Delta)$-expander, and furthermore, $|\tilde{U}_i| \leq n/2$, so $|V(H_i)| \geq n/2$. Denote $\varphi^* = \frac{\varphi}{6\Delta}$, and note that:

$$k' = \left\lfloor \frac{n^{1-20\epsilon}\varphi^2}{2^{11} \cdot \Delta^4} \right\rfloor \leq \frac{|V(H_i)|^{1-20\epsilon}(\varphi^*)^2}{\Delta^2}$$

Therefore, we can apply the algorithm from Lemma 8.1 to graph H_i with the sequence Σ_i of edge deletions. We denote by U_i the incremental set of vertices of H_i that the algorithm maintains. The set U of vertices of G that our algorithm maintains over the course of the ith phase is defined to be $\tilde{U}_i \cup U_i$.

Recall that the total number of edges that are deleted over the course of the first $(i-1)$ phases of the algorithm from graph G is $(i-1) \cdot k' \geq (i-1) \cdot \left\lfloor \frac{n^{1-20\epsilon}\varphi^2}{2^{11} \cdot \Delta^4} \right\rfloor$.

Since the total number of edges deleted from graph G' over the course of the first $(i-1)$ phases is bounded by $(i-1) \cdot \frac{n^{1-20\epsilon} \cdot \varphi}{32 \cdot \Delta}$, from Theorem 3.4, we get that:

$$|\tilde{U}_i| \leq \frac{n^{1-20\epsilon} \cdot (i-1)}{4}.$$

Consider some integer $0 \leq t \leq k'$, and the time during the execution of phase i, immediately after the t-th edge of Σ_i is deleted from graph G in phase i. Then at this point, the total number of edges deleted from graph G since the beginning of the algorithm is at least:

$$m_t^i = (i-1) \cdot \left\lfloor \frac{n^{1-20\epsilon}\varphi^2}{2^{11} \cdot \Delta^4} \right\rfloor + t.$$

At the same time:

$$|U| = |\tilde{U}_i| + |U_i| \leq (i-1) \cdot \frac{n^{1-20\epsilon}}{4} + \frac{4\Delta t}{\varphi^*} = (i-1) \cdot \frac{n^{1-20\epsilon}}{4} + \frac{24\Delta^2 t}{\varphi} \leq \frac{2^{11}\Delta^4}{\varphi^2} \cdot m_t^i.$$

Notice that, over the course of the ith phase, $V(H_i) \setminus U_i = V(G) \setminus (U_i \cup \tilde{U}_i) = V(G) \setminus U$ holds. Therefore, when short-path query arrives for a pair $x, y \in V(G) \setminus U$ of vertices, it must be the case that $x, y \in V(H_i) \setminus U_i$. We can then perform short-path query in the data structure maintained by the algorithm from Lemma 8.1, to obtain a path P in the current graph $H_i \subseteq G$, of length at most $\frac{2^{O(1/\epsilon^6)} \cdot \Delta \cdot \log n}{\varphi^*} \leq \frac{2^{O(1/\epsilon^6)} \cdot \Delta^2 \cdot \log n}{\varphi}$, in time $O(|E(P)|)$. We return this path as the response to the query.

From the above discussion, at every time t during the execution of phase i, if m_t^i is the number of edges deleted so far by the algorithm, and $U^{(t)}$ is the current set U, then:

$$|U^{(t)}| \leq \frac{2^{11}\Delta^2}{\varphi^2} \cdot m_t^i.$$

Once all edges of Σ_i edges are deleted from graph G, we let $\tilde{E}_i$ be the set of all edges that are incident to the vertices in the current set U_i. Observe that:

$$|\tilde{E}_i| \leq \Delta \cdot |U_i| \leq \frac{4\Delta^2}{\varphi^*} \cdot k' \leq \frac{24\Delta^3}{\varphi} \cdot \frac{n^{1-20\epsilon}\varphi^2}{2^{11} \cdot \Delta^4} \leq \frac{n^{1-20\epsilon} \cdot \varphi}{64 \cdot \Delta}.$$

since $\varphi^* = \frac{\varphi}{6\Delta}$ and $k' = \left\lfloor \frac{n^{1-20\epsilon}\varphi^2}{2^{11} \cdot \Delta^4} \right\rfloor$.

We delete the edges of $E_i \cup \tilde{E}_i$ from graph G', and update the data structure maintained by the algorithm from Theorem 3.4. Since $|E_i| \leq k' = \left\lfloor \frac{n^{1-20\epsilon}\varphi^2}{2^{11} \cdot \Delta^4} \right\rfloor$, we get that $|E_i \cup \tilde{E}_i| \leq \frac{n^{1-20\epsilon} \cdot \varphi}{32 \cdot \Delta}$. Recall that we have assumed that, over the course of the first $(i-1)$ phases, the total number of edges deleted from graph G' is bounded by $(i-1) \cdot \frac{n^{1-20\epsilon} \cdot \varphi}{32 \cdot \Delta}$. We then get that, over the course of the first i phases, the total number of edges deleted from graph G' is bounded by $i \cdot \frac{n^{1-20\epsilon} \cdot \varphi}{32 \cdot \Delta}$. We then let $\tilde{U}_{i+1}$ the set $\tilde{U}$ that the algorithm from Theorem 3.4 maintains after the edges of $E_i \cup \tilde{E}_i$ are deleted from G'. Since the vertices from the set U_i obtained at the end of phase i are isolated in graph $G' \setminus \tilde{E}_i$, while graph $G' \setminus \tilde{U}_{i+1}$ must be a $\varphi/(6\Delta)$-expander, we get that $U_i \subseteq \tilde{U}_{i+1}$. We then let the set U that the algorithm maintains be $\tilde{U}_{i+1}$, and we continue to the next phase, with the graph $H_{i+1} = G' \setminus \tilde{U}_{i+1}$.

It is easy to verify that the set U of vertices that the algorithm maintains is incremental. Indeed, consider some phase i of the algorithm. Throughout the phase, we let $U = \tilde{U}_i \cup U_i$, where $\tilde{U}_i$ is fixed over the course of the phase, and U_i is incremental. If we denote by U_i' the set U_i at the end of Phase i, then we are guaranteed that $U_i' \subseteq \tilde{U}_{i+1}$, and, from Theorem 3.4, $\tilde{U}_i \subseteq \tilde{U}_{i+1}$. At the beginning of Phase $(i+1)$, we set $U = \tilde{U}_{i+1}$, and so $\tilde{U}_i \cup U_i' \subseteq U$ at this point. Therefore, set U is incremental throughout the algorithm.

It now only remains to bound the total update time of the algorithm. The algorithm consists of at most $O(n^{20\epsilon})$ phases. In every phase, we run the algorithm from Lemma 8.1, whose total update time is $O\left(\frac{m^{1+O(\epsilon)} \cdot \Delta^3}{(\varphi^*)^2}\right) \leq O\left(\frac{m^{1+O(\epsilon)} \cdot \Delta^5}{\varphi^2}\right)$.

Additionally, the total update time of the algorithm from Theorem 3.4, over the course of at most m deletions of edges from G', is bounded by $\widetilde{O}\left(\frac{m\Delta^2}{\varphi^2}\right)$.

Altogether, the total update time of the algorithm is bounded by:

$$O(n^{20\epsilon}) \cdot O\left(\frac{m^{1+O(\epsilon)} \cdot \Delta^5}{\varphi^2}\right) + \widetilde{O}\left(\frac{m\Delta^2}{\varphi^2}\right) \leq O\left(\frac{m^{1+O(\epsilon)} \cdot \Delta^5}{\varphi^2}\right).$$

In order to complete the proof of Theorem 2.4, it now remains to prove Lemma 8.1.

8.1 Proof of Lemma 8.1

We start by describing the data structures that the algorithm maintains, together with their initialization. We then describe an algorithm for maintaining the data structures under the deletion of edges from G. Finally, we describe an algorithm for responding to short-path query.

Before we do so, we establish some bounds on the parameters that will be useful for us later. All of these bounds follow from the fact that $\frac{2}{(\log n)^{1/12}} < \epsilon < \frac{1}{400}$ holds, from the statement of Lemma 8.1.

First, since $\epsilon > \frac{2}{(\log n)^{1/12}}$, we get that $\log n \geq (2/\epsilon)^{12}$, and:

$$n \geq 2^{(2/\epsilon)^{12}} \geq 2^{800^{12}}. \tag{11}$$

Additionally:

$$n^{\epsilon^8} > n^{(2/(\log n)^{1/12})^8} = n^{256/(\log n)^{2/3}} > 2^{(\log n)^{1/3}} > \log n. \tag{12}$$

8.1.1 Data Structures and Initialization

We start by applying the algorithm from Corollary 5.3 to graph G, with the set $T = V(G)$ of terminals, parameter ϵ given by the statement of Lemma 8.1, and parameters $d = \frac{64\Delta \log m}{\varphi^*}$ and $\eta = \frac{512\Delta^2 \log m}{(\varphi^*)^2}$.

We claim that the algorithm may not return a pair $T_1, T_2 \subseteq T$ of disjoint subsets of terminals, and a set E' of edges of G, with $|T_1| = |T_2|$, $|T_1| \geq \frac{n^{1-4\epsilon^3}}{4}$ and $|E'| \leq \frac{d \cdot |T_1|}{\eta}$, such that for every pair $t \in T_1, t' \in T_2$ of terminals, $\mathsf{dist}_{G \setminus E'}(t, t') > d$. Indeed, assume for contradiction that the algorithm returns a pair T_1, T_2 of disjoint subsets of vertices of G with the above properties. Denote $\varphi = \frac{\varphi^*}{\Delta}$, and note that $|E'| \leq \frac{d \cdot |T_1|}{\eta} = \frac{\varphi^* \cdot |T_1|}{8\Delta} \leq \frac{\varphi \cdot |T_1|}{4}$. Note also that $d = \frac{64\Delta \log m}{\varphi^*} \geq \frac{32 \log m}{\varphi}$. Clearly, we can view (T_1, T_2, E') as a (δ, d)-distancing in graph G, for some parameter $0 < \delta < 1$. From Lemma 4.1, there is a cut (X, Y) in graph G, with $T_1 \subseteq X$ and $T_2 \subseteq Y$, such that $|E_G(X, Y)| \leq \varphi \cdot \min\{|E(X)|, |E(Y)|\}$. Since the maximum vertex degree in G is bounded by Δ, $|E(X)| < \Delta \cdot |X|$, and similarly $|E(Y)| < \Delta \cdot |Y|$. Therefore, we are guaranteed that $|E(X, Y)| < \varphi \cdot \Delta \cdot \min\{|X|, |Y|\} = \varphi^* \cdot \min\{|X|, |Y|\}$, contradicting the fact that G is a φ^*-expander.

We conclude that the algorithm from Corollary 5.3 must return a graph H with $V(H) \subseteq V(G)$, $|V(H)| = N^{1/\epsilon} \geq n - n^{1-\epsilon/2} \geq n/2$, where $N = \lfloor n^\epsilon \rfloor$, so that the maximum vertex degree in H is at most $n^{32\epsilon^3}$. The algorithm also must return an embedding $\mathcal{P}$ of H into G via paths of length at most d that cause congestion at most $\eta \cdot n^{32\epsilon^3}$, and a level-$(1/\epsilon)$ hierarchical support structure for H, such that H is $(\eta', \tilde{d})$-well-connected with respect to the set $S(H)$ of vertices defined by the support

structure, where $\eta' = N^{6+256\epsilon}$, and $\tilde{d} = 2^{c/\epsilon^5}$, with c being the constant used in the definition of the Hierarchical Support Structure. Recall that the running time of the algorithm is:

$$O\left(n^{1+O(\epsilon)} + |E(G)| \cdot n^{O(\epsilon^3)} \cdot (\eta + d\log n)\right) \leq O\left(n^{1+O(\epsilon)} + n^{1+O(\epsilon^3)} \cdot \frac{\Delta^3 \log n}{(\varphi^*)^2}\right) \leq O\left(\frac{n^{1+O(\epsilon)} \cdot \Delta^3}{(\varphi^*)^2}\right).$$

(We have used the fact that $d = \frac{64\Delta \log m}{\varphi^*}$, $\eta = \frac{512\Delta^2 \log m}{(\varphi^*)^2}$, and Inequality 12).

For every edge $e' \in E(H)$, let $P(e') \in \mathcal{P}$ be the path embedding edge e' into G. For every edge $e \in E(G)$, we will maintain a list $L(e)$ of all edges $e' \in E(H)$ with $e \in E(P(e'))$. The list also contains, for each edge $e' \in L(e)$, a pointer to edge e' in graph H, and each edge $e' \in E(H)$ maintains a pointer to every edge in $E(P(e'))$. Whenever an edge $e \in E(G)$ is deleted from graph G, we will delete every edge $e' \in L(e)$ from graph H. Since the paths in $\mathcal{P}$ cause congestion at most $\eta \cdot n^{32\epsilon^3} = \frac{512\Delta^2 \cdot n^{32\epsilon^3} \log m}{(\varphi^*)^2} \leq \frac{\Delta^2 \cdot n^{33\epsilon^3}}{(\varphi^*)^2}$ in H, every deletion of an edge in G may trigger the deletion of at most $\tilde{\eta} = \frac{\Delta^2 \cdot n^{33\epsilon^3}}{(\varphi^*)^2}$ edges from H. Let $q = 1/\epsilon$. Then the total number of edge deletions from graph H over the course of the algorithm is bounded by:

$$\frac{n^{1-20\epsilon}(\varphi^*)^2}{\Delta^2} \cdot \tilde{\eta} = n^{1-20\epsilon+33\epsilon^2} \cdot 512\log m \leq \frac{n^{1-16\epsilon}}{2} \leq N^{q-16q\epsilon} < N^{q-8-300q\epsilon^2},$$

as $q = 1/\epsilon$ and $\frac{n}{2} \leq N^q$.

Note that $\eta' = N^{6+256\epsilon} = N^{6+256q\epsilon^2} = \eta_q$ and $\tilde{d} = 2^{c/\epsilon^5} = \tilde{d}_q$, where η_q and $\tilde{d}_q$ are the parameters from the definition of Hierarchical Support Structure. We will use the algorithm from Theorem 7.1 in graph H, with parameter $j = q$, and parameters ϵ, N remaining unchanged. In order to be able to use the theorem, we need to verify that $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^6}$ holds. Since $N = \lfloor n^\epsilon \rfloor$, we get that:

$$\frac{N^{\epsilon^4}}{\log N} \geq \frac{n^{\epsilon^6}}{\epsilon \cdot \log n} \geq n^{\epsilon^6/2} \geq 2^{128/\epsilon^6}.$$

(we have used the fact that, from Inequality 12, $\log n < n^{\epsilon^8}$, from Inequality 11, $n \geq 2^{(2/\epsilon)^{12}}$, and $\epsilon < 1/400$).

As observed already, the number of edge deletions that graph H undergoes over the course of the algorithm is bounded less than $N^{q-8-300q\epsilon^2} = \Lambda_q$. We will maintain a data structure from Theorem 7.1 in graph H, with parameters ϵ, N and q as defined above. We denote this data structure by $\mathcal{D}(H)$. At the beginning of the algorithm, we initialize this data structure. Recall that data structure $\mathcal{D}(H)$ maintains a decremental set $S'(H) \subseteq S(H)$ of vertices of H, called supported vertices, such that, at the beginning of the algorithm, $S'(H) = S(H)$. The algorithm ensures that $|S'(H)| \geq \frac{N^q}{16^q}$ holds over the course of the algorithm, and it supports short-path queries between supported vertices: given a pair $x, y \in S'(H)$ of vertices, return a path P connecting x to y in the current graph H, whose length is at most $d^*_q = 2^{O(q/\epsilon^5)} = 2^{O(1/\epsilon^6)}$, in time $O(|E(P)|)$. If $m' \leq n^{1+32\epsilon^3}$ is the number of edges in H at the beginning of the algorithm, then the total update time needed to maintain data structure $\mathcal{D}(H)$ is bounded by:

$$O\left(qN^{q+3} \cdot 2^{O(1/\epsilon^6)} + m' \cdot N^2 \cdot 2^{O(1/\epsilon^6)}\right) \leq O\left(n^{1+O(\epsilon)} \cdot 2^{O(1/\epsilon^5)}\right) \leq O\left(n^{1+O(\epsilon)}\right)$$

(We have used the fact that, from Inequality 11, $n^\epsilon \geq 2^{(2/\epsilon)^{12} \cdot \epsilon} \geq 2^{O(1/\epsilon^6)}$).

Lastly, we maintain an ES-Tree data structure τ in graph G, rooted at the set $S'(H)$ of vertices, with depth parameter $d = \frac{64\Delta \log m}{\varphi^*}$. Specifically, we maintain a graph G', that is obtained from graph G by adding a source vertex s, that connects to every vertex that lies in the current set $S'(H)$ of supported vertices with an edge. We then let τ be an ES-Tree data structure in graph G', rooted at s, with depth $d+1$. Whenever a vertex x is deleted from set $S'(H)$, we will delete edge (s,x) from G', and update the data structure τ accordingly. Also, whenever an edge e is deleted from graph G, we will also delete e from graph G', and update the data structure τ accordingly. Throughout the algorithm, we let U be the set of all vertices $v \in V(G)$, such that $v \notin V(\tau)$. In other words, $\mathsf{dist}_G(S'(H), v) > d$. Clearly, the set U of vertices is incremental: vertices can joint it but they cannot leave it. In the next claim we bound the cardinality of U.

Claim 8.2 *Let t be any time during the algorithm's execution, let E'_t be the set of edges that were deleted so far from G, and let U_t be the current set U of vertices. Then $|U_t| \leq \frac{4\Delta|E'_t|}{\varphi^*}$.*

Proof: Assume otherwise, and let t be some time during the algorithm's execution, when $|U_t| > \frac{4\Delta|E'_t|}{\varphi^*}$ holds. Denote $\varphi = \frac{\varphi^*}{\Delta}$, so that $|E'_t| < \varphi \cdot |U_t|/4$.

Let S'_t be the set $S'(H)$ of vertices at time t. Recall that we are guaranteed that $|S'(H)| \geq \frac{N^q}{16^q} \geq \frac{n}{2^{4/\epsilon+1}}$ holds throughout the algorithm. Since the total number of the edges deleted from G over the course of the algorithm is at most $\frac{n^{1-20\epsilon}(\varphi^*)^2}{\Delta^2} = n^{1-20\epsilon}\varphi^2 \leq n^{1-20\epsilon} \cdot \varphi$, we get that $|E'_t| \leq n^{1-20\epsilon} \cdot \varphi$. On the other hand, from Inequality 11, $n^{20\epsilon} \geq 2^{4/\epsilon+4}$, and so:

$$|S'_t| \geq \frac{n}{2^{4/\epsilon+1}} \geq \frac{n^{1-20\epsilon}}{4} \geq \frac{4|E'_t|}{\varphi}.$$

We conclude that $|E'_t| \leq \frac{\varphi}{4} \cdot \min\{|S'_t|, |U_t|\}$. Denote $M = \min\{|S'_t|, |U_t|\}$. Let $X \subseteq S'_t$, $Y \subseteq U_t$ be arbitrary subsets of vertices of cardinality M each. Let G^0 be the graph G at the beginning of the algorithm. Clearly, $\mathsf{dist}_{G^0 \setminus E'_t}(X,Y) > d$. Recall that $d = \frac{64\Delta \log m}{\varphi^*} = \frac{64 \log m}{\varphi}$. From Lemma 4.1, there is a cut (X', Y') in graph G^0, with with $X \subseteq X'$ and $Y \subseteq Y'$, such that $|E_{G^0}(X', Y')| \leq \varphi \cdot \min\{|E_{G^0}(X')|, |E_{G^0}(Y')|\}$. Since $|E_{G^0}(X')| < \Delta \cdot |X'|$ and $|E_{G^0}(Y')| < \Delta \cdot |Y'|$, we get that:

$$|E_{G^0}(X', Y')| < \varphi \cdot \Delta \cdot \min\left\{|X'|, |Y'|\right\} = \varphi^* \cdot \min\left\{|X'|, |Y'|\right\},$$

contradicting the fact that graph G^0 is a φ^*-expander. □

At the beginning of the algorithm, we initialize the ES-Tree data structure τ, and set $U = \emptyset$. The total update time needed in order to maintain τ is bounded by $O(md \log m) \leq O\left(\frac{m\Delta \log^2 m}{\varphi^*}\right)$.

We now bound the running time that is needed to initialize all data structures, and to maintain data structures $\mathcal{D}(H)$, τ, and $\{L(e)\}_{e \in E(G)}$ over the course of the algorithm. The running time of the algorithm from Corollary 5.3, from the above discussion, is bounded by $O\left(\frac{n^{1+O(\epsilon)} \cdot \Delta^3}{(\varphi^*)^2}\right)$. The time that is needed in order to initialize and maintain the lists $L(e)$ for edges $e \in E(G)$ is subsumed by this running time. Additionally, from the above discussion, the total update time of data structure $\mathcal{D}(H)$ is bounded by $O\left(n^{1+O(\epsilon)}\right)$, and the total update time of data structure τ is bounded by $O\left(\frac{m\Delta \log^2 m}{\varphi^*}\right)$. The set U of vertices can be maintained within this asymptotic running time. Overall, the total time needed to initialize all data structures, and to maintain data structures $\mathcal{D}(H)$, τ, and $\{L(e)\}_{e \in E(G)}$ over the course of the algorithm is bounded by:

ALGORITHM DeleteExpanderEdge(e)

1. For every edge $e' \in L(e)$ do:
   (a) Delete e' from H and update data structure $\mathcal{D}(H)$. Let X be the set of vertices that were deleted from $S'(H)$ as the result of this update.
   (b) For every edge $e'' \in E(G)$ with $e' \in L(e'')$, delete e' from $L(e'')$.
   (c) for every vertex $x \in X$, delete edge (s, x) from graph G' and update the ES-Tree τ with this deletion. Add every vertex that is removed from τ to U.
2. Delete edge e from graph G' and update data structure τ accordingly. Add every vertex that is removed from τ to U.

Figure 2: Algorithm DeleteExpanderEdge(e)

$$O\left(\frac{m\Delta \log^2 m}{\varphi^*}\right) + O\left(\frac{n^{1+O(\epsilon)} \cdot \Delta^3}{(\varphi^*)^2}\right) \leq O\left(\frac{n^{1+O(\epsilon)} \cdot \Delta^3}{(\varphi^*)^2}\right).$$

8.1.2 Maintaining the Data Structures

Maintaining the data structures under the deletion of edges from G is now straightforward. In Figure 2 we describe algorithm DeleteExpanderEdge(e) that is invoked whenever an edge e is deleted from graph G. We start by considering every edge $e' \in E(H)$ whose embedding path $P(e')$ contains edge e – in other words, all edges of $L(e)$. We delete each such edge e' from graph H, and update data structure $\mathcal{D}(H)$ with this deletion. As a result, it is possible that some vertices are removed from set $S'(H)$. For each such vertex x, we delete edge (s, x) from graph G', and update the data structure τ accordingly. Finally, we delete edge e from data structure τ. Whenever a vertex leaves the tree τ, we add it to U.

It is easy to verify that the total update time of the algorithm is dominated by the time needed to initialize the data structures, and to maintain data structures $\mathcal{D}(H)$, τ, and $\{L(e)\}_{e \in E(G)}$. From the above discussion, the total update time of the algorithm is bounded by $O\left(\frac{n^{1+O(\epsilon)} \cdot \Delta^3}{(\varphi^*)^2}\right)$.

8.1.3 Responding to Short-Path Queries

We assume that we are given a pair of vertices $x, y \in V(G) \setminus U$, and describe an algorithm for responding to short-path query between x and y. Recall that our goal is to return an x-y path P in the current graph G, of length at most $\frac{2^{O(1/\epsilon^6)} \cdot \Delta \cdot \log n}{\varphi^*}$, with query time $O(|E(P)|)$.

Using the ES-Tree τ, we compute a path Q connecting x to some vertex $x' \in S'(H)$, and a path Q' connecting vertex y to some vertex $y' \in S'(H)$, so that the length of each path is bounded by d. Next, we query data structure $\mathcal{D}(H)$ with the pair $x', y' \in S'(H)$ of vertices. The data structure must return a path $\tilde{Q}$ connecting x' to y' in H, whose length is at most $2^{O(1/\epsilon^6)}$, in time $O(|E(\tilde{Q})|)$. Using the embedding $\mathcal{P}$ of H into G, in which the length of every path is bounded by d, we can compute a path Q'' in graph G, connecting x' to y', whose length is bounded by $|E(\tilde{Q})| \cdot d \leq 2^{O(1/\epsilon^6)} \cdot d$. Lastly, by concatenating the paths Q, Q'' and Q', we obtain a path P connecting x to y in graph G, whose length is at most:

$$2^{O(1/\epsilon^6)} \cdot d \leq \frac{2^{O(1/\epsilon^6)} \cdot \Delta \cdot \log m}{\varphi^*} \leq \frac{2^{O(1/\epsilon^6)} \cdot \Delta \cdot \log n}{\varphi^*}.$$

It is easy to see that the running time of the algorithm is $O(|E(P)|)$.

9 Advanced Path Peeling – Proof of Theorem 2.5

In this section we prove Theorem 2.5. The main tool in the proof is the following theorem.

Theorem 9.1 *There is a large enough constant c^*, and a deterministic algorithm, whose input consists of a connected n-vertex m-edge graph G, a collection $M = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of pairs of vertices of G, such that M is a matching, and parameters $d, \eta > 0$, $0 < \alpha \leq 1/2$ and $\frac{2}{(\log n)^{1/24}} < \epsilon < \frac{1}{400}$, such that $1/\epsilon$ is an integer and $256d < \eta \leq \frac{d^2}{2^{c^*/\epsilon^6} \cdot \log m}$ holds. The algorithm computes one of the following:*

- *either a cut (A, B) with $|E_G(A, B)| \leq \frac{1024d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$, and each of A, B contains at least $\frac{\alpha k}{8}$ vertices of set $T = \{s_1, t_1, \ldots, s_k, t_k\}$; or*
- *a routing $\mathcal{P}$ in G of a subset $M' \subseteq M$ containing at least $(1-\alpha)k$ pairs of vertices, such that every path in $\mathcal{P}$ has length at most d, and the total congestion caused by the paths in $\mathcal{P}$ is at most $\frac{4\eta}{\alpha}$.*

The running time of the algorithm is bounded by $O\left(m^{1+O(\epsilon)}(d^2 + \eta d)\right)$.

The proof of Theorem 2.5 easily follows from Theorem 9.1. Let $z = \left\lceil \frac{1}{\epsilon} \right\rceil$, and let $\epsilon' = \frac{1}{z}$. Clearly, $\frac{1}{\epsilon} \leq z \leq \frac{2}{\epsilon}$, and so $\frac{\epsilon}{2} \leq \epsilon' \leq \epsilon$. Since we have assumed that $\frac{4}{(\log n)^{1/24}} < \epsilon < \frac{1}{400}$, we get that $\frac{2}{(\log n)^{1/24}} < \epsilon' < \frac{1}{400}$. For convenience, in the remainder of the proof we will denote ϵ' by ϵ.

Let c^* be the constant from Theorem 9.1. We set $d = \frac{2^{c^*/\epsilon^6+10} \cdot \log m}{\varphi}$ and $\eta = \frac{1024d}{\varphi} = \frac{2^{c^*/\epsilon^6+20} \cdot \log m}{\varphi^2}$. It is immediate to verify that $\eta > 256d$. Observe also that:

$$\frac{d^2}{2^{c^*/\epsilon^6} \cdot \log m} = \frac{1024d}{\varphi} \geq \eta.$$

We start by considering the case where $\frac{1}{\alpha} < \log n$. In this case, we apply the algorithm from Theorem 9.1 to graph G, the set M of pairs of its vertices, and parameters d, η, α and ϵ. Assume first that the outcome of the algorithm is a cut (A, B) with $|E_G(A, B)| \leq \frac{1024d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\} = \varphi \cdot \min\{|E_G(A)|, |E_G(B)|\}$, and $|T \cap A|, |T \cap B| \geq \frac{\alpha k}{8}$. In this case, we return the cut (A, B) as the outcome of the algorithm. Otherwise, the outcome of the algorithm from Theorem 9.1 is a routing $\mathcal{P}$ in G of a subset $M' \subseteq M$ containing at least $k \cdot (1-\alpha)$ pairs of vertices, such that every path in $\mathcal{P}$ has length at most $d \leq \frac{2^{O(1/\epsilon^6)} \cdot \log n}{\varphi}$, and the total congestion caused by the paths in $\mathcal{P}$ is at most $\frac{4\eta}{\alpha} \leq \frac{2^{O(1/\epsilon^5)} \cdot \log n}{\alpha \cdot \varphi^2}$. We then return this set of paths as the outcome of the algorithm.
The running time of the algorithm from Theorem 9.1 is bounded by:

$$O\left(m^{1+O(\epsilon)}(d^2 + \eta d)\right) \leq O\left(m^{1+O(\epsilon)}\left(\frac{2^{O(1/\epsilon^6)} \cdot \log^2 n}{\varphi^2} + \frac{2^{O(1/\epsilon^6)} \cdot \log^2 n}{\varphi^3}\right)\right) \leq O\left(\frac{m^{1+O(\epsilon)}}{\varphi^3}\right).$$

Next, we consider the case where $\frac{1}{\alpha} > \log n$. In this case, we perform at most $\log n$ iterations. At the beginning of iteration i, we are given a collection $M_i \subseteq M$ of pairs of vertices that have been already routed, together with their routing $\mathcal{P}_i$ in graph G. At the beginning of the algorithm, $M_1 = \emptyset$ and $\mathcal{P}_1 = \emptyset$. The iterations continue as long as $|M_i| < (1-\alpha)k$ holds.

We now describe the execution of the ith iteration. Let $M_i' = M \setminus M_i$, and recall that $|M_i'| \geq \alpha k$ must hold. We apply the algorithm from Theorem 9.1 to graph G, the set M_i' of pairs of its vertices, and parameters $d, \eta, \alpha' = 1/2$ and ϵ. Assume first that the outcome of the algorithm is a cut (A,B) with $|E_G(A,B)| \leq \frac{1024d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\} = \varphi \cdot \min\{|E_G(A)|, |E_G(B)|\}$, and $|T \cap A|, |T \cap B| \geq \frac{|M_i'|}{16} \geq \frac{\alpha k}{16}$. In this case, we terminate the algorithm and return the cut (A,B) as the outcome of the algorithm. Otherwise, the outcome of the algorithm from Theorem 9.1 is a routing $\tilde{\mathcal{P}}_i$ in G of a subset $\tilde{M}_i \subseteq M_i'$ containing at least $|M_i'|/2$ pairs of vertices, such that every path in $\mathcal{P}$ has length at most $d \leq \frac{2^{O(1/\epsilon^6)} \cdot \log n}{\varphi}$, and the total congestion caused by the paths in $\mathcal{P}$ is at most $8\eta \leq \frac{2^{O(1/\epsilon^6)} \cdot \log n}{\varphi^2}$. We then set $M_{i+1} = M_i \cup \tilde{M}_i$, $\mathcal{P}_{i+1} = \mathcal{P}_i \cup \tilde{\mathcal{P}}_i$, and continue to the next iteration.

Assume that the last iteration of the algorithm is iteration i. If the algorithm did not terminate with a cut, then $|M_{i+1}| \geq (1-\alpha)k$ must hold. We then return the set $M' = M_{i+1}$ of pairs of vertices, and their routing $\mathcal{P} = \mathcal{P}_{i+1}$. It is easy to verify that the cardinality of the set $M_{i'}'$ of pairs of vertices that remains to be routed decreases by at least factor 2 in every iteration, and so the number of iterations in the algorithm is bounded by $\log k$. Since, for every iteration i', the congestion caused by the set $\tilde{\mathcal{P}}_{i'}$ of paths is at most $\frac{2^{O(1/\epsilon^6)} \cdot \log n}{\varphi^2}$, the total congestion caused by the set $\mathcal{P}$ of paths is at most $\frac{2^{O(1/\epsilon^6)} \cdot \log^2 n}{\varphi^2}$. The running time of a single iteration is bounded by $O\left(\frac{m^{1+O(\epsilon)}}{\varphi^3}\right)$ as before, and, since the number of iterations is $O(\log n)$, the total running time of the algorithm remains bounded by $O\left(\frac{m^{1+O(\epsilon)}}{\varphi^3}\right)$.

In the remainder of this section we prove Theorem 9.1. Following is a key lemma that we use in the proof.

Lemma 9.2 *There is a large enough constant c^*, and a deterministic algorithm, whose input consists of a connected m-edge graph G with $|V(G)| \leq n$, a collection $M = \{(s_1,t_1), \ldots, (s_k,t_k)\}$ of pairs of vertices of G, such that M is a matching, and parameters $d, \eta > 0$ and $\frac{2}{(\log n)^{1/24}} < \epsilon < \frac{1}{400}$, such that $1/\epsilon$ is an integer, and $128d < \eta \leq \frac{d^2}{2^{c^*/\epsilon^6} \cdot \log m}$ holds. The algorithm computes one of the following:*

- *either a cut (A,B) with $|E_G(A,B)| \leq \frac{64d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$, and each of A, B contains at least $\frac{k^{1-\epsilon}}{16}$ vertices of set $T = \{s_1, t_1, \ldots, s_k, t_k\}$; or*
- *a routing $\mathcal{P}$ in G of a subset $M' \subseteq M$ containing at least $z = \frac{k^{1-22\epsilon}}{d}$ pairs of vertices, such that every path in $\mathcal{P}$ has length at most d, and the total congestion caused by the paths in $\mathcal{P}$ is at most η.*

The running time of the algorithm is bounded by $O\left(m^{1+O(\epsilon)}(\eta + d\log n)\right)$.

We defer the proof of Lemma 9.2 to Section 9.1, after we complete the proof of Theorem 9.1 using it. Our algorithm iteratively applies the algorithm from Lemma 9.2, while gradually constructing both a routing of some pairs from M, and a low-conductance cut in G.

Let $\eta' = \frac{4\eta}{\alpha}$. Our algorithm consists of two stages. In the first stage, we either construct the desired routing $\mathcal{P}$ of a large subset $M' \subseteq M$ of pairs of vertices, or compute a collection $\mathcal{S}$ of disjoint subsets of vertices of G with some useful properties. In the former case, we terminate the algorithm and

return the resulting routing $\mathcal{P}$, while in the latter case we continue to Stage 2, in which we exploit the collection $\mathcal{S}$ of vertex subsets, in order to construct the desired low-conductance cut (A, B). We now describe each of the two stages in turn.

Stage 1: Constructing a Routing

Let $q' = 4d \cdot k^{22\epsilon} \cdot \log k$. Our algorithm in Stage 1 consists of at most q' iterations. At the beginning of iteration q, we are given a subset $M'_q \subseteq M$ of pairs of vertices with $|M'_q| < (1-\alpha) \cdot k$, and a routing $\mathcal{P}_q$ of the matching M'_q in graph G. Additionally, we are given a collection $\mathcal{S}_q$ of disjoint subsets of vertices of G, and we denote $A_q = \bigcup_{S \in \mathcal{S}_q} S$. We will ensure that the following invariants hold:

I1. every path in $\mathcal{P}_q$ has length at most d;

I2. $|\mathcal{P}_q| < (1-\alpha)k$;

I3. the paths in $\mathcal{P}_q$ cause congestion at most η' in G;

I4. if we denote by E'_q the set of all edges that lie on at least $\eta'/2$ paths in $\mathcal{P}_q$, then $\left|\left(\bigcup_{S \in \mathcal{S}_q} \delta_G(S)\right) \setminus E'_q\right| \leq \frac{64d}{\eta} \sum_{S \in \mathcal{S}_q} |E_G(S)|$; and

I5. $|A_q \cap T| < \frac{\alpha k}{2}$.

At the beginning of the algorithm, we set $\mathcal{P}_1 = \emptyset$ and $\mathcal{S}_1 = \emptyset$, so $A_1 = \emptyset$ holds. Clearly, all invariants hold for this setting. We now describe the execution of the qth iteration, for some $q \geq 1$. We assume that we are given a set $M'_q \subseteq M$ of pairs of vertices, a set $\mathcal{P}_q$ of paths routing the pairs in M'_q in G, and a collection $\mathcal{S}_q$ of disjoint subsets of vertices of G, for which invariants I1–I5 hold.

We let M_q be a set of pairs of vertices of G, containing all pairs $(s_i, t_i) \in M \setminus M'_q$ with $s_i, t_i \in V(G) \setminus A_q$. We also let G_q be the graph obtained from G, after we delete from it all vertices of A_q, and all edges $e \in E(G)$, such that e that belongs to at least $\eta'/2$ paths of $\mathcal{P}_q$. In other words, $G_q = (G \setminus A_q) \setminus E'_q$. Denote $k_q = |M_q|$. Since, from Invariant I5, $|A_q \cap T| < \frac{\alpha k}{2}$, while from Invariant I2, $|M \setminus M'_q| \geq k - |\mathcal{P}_q| \geq \alpha k$, we get that $k_q > \frac{\alpha k}{2}$ must hold. Notice that graph G_q may not be connected. We assume first that there is some connected component C_q of graph G_q, and a subset $\tilde{M}_q \subseteq M_q$ containing at least $k_q/2$ pairs, such that all vertices participating in the pairs in $\tilde{M}_q$ lie in C_q.

We apply the algorithm from Lemma 9.2 to graph C_q, the set $\tilde{M}_q$ of pairs of vertices, and parameters d, η, ϵ that remain unchanged. We now consider two cases. The first case happens if the algorithm from Lemma 9.2 returns a cut (X_q, Y_q) in graph C_q, with $|E_{G_q}(X_q, Y_q)| \leq \frac{64d}{\eta} \cdot \min\left\{|E_{G_q}(X_q)|, |E_{G_q}(Y_q)|\right\} \leq \frac{64d}{\eta} \cdot \min\left\{|E_G(X_q)|, |E_G(Y_q)|\right\}$. Recall that we are also guaranteed that each of X_q, Y_q contains at least $\frac{k_q^{1-\epsilon}}{32}$ vertices of set $T' = \left\{s_i, t_i \mid (s_i, t_i) \in \tilde{M}_q\right\}$. We say that iteration q is a *type-1 iteration*. We assume w.l.o.g. that $|X_q \cap T'| \leq |Y_q \cap T'|$. We set $S_{q+1} = S_q \cup \{X_q\}$, and we let $M'_{q+1} = M'_q$ and $\mathcal{P}_{q+1} = \mathcal{P}_q$. It is immediate to verify that Invariants I1–I3 continue to hold for $\mathcal{P}_{q+1}$. It is also immediate to verify that $E'_{q+1} = E'_q$. Therefore, $\left|\left(\bigcup_{S \in \mathcal{S}_q} \delta_G(S)\right) \setminus E'_{q+1}\right| \leq \frac{64d}{\eta} \sum_{S \in \mathcal{S}_q} |E_G(S)|$. Let $E^*_q = \left(\left(\bigcup_{S \in \mathcal{S}_{q+1}} \delta_G(S)\right) \setminus \left(\bigcup_{S \in \mathcal{S}_q} \delta_G(S)\right)\right) \setminus E'_q$. Then $E^*_q = E_{G_q}(X_q, Y_q)$, and we are guaranteed that $|E^*_q| \leq \frac{64d}{\eta} \cdot |E_G(X_q)|$. Therefore, we get that:

$$\left|\left(\bigcup_{S\in\mathcal{S}_{q+1}}\delta_G(S)\right)\setminus E'_{q+1}\right| \leq \left|\left(\bigcup_{S\in\mathcal{S}_q}\delta_G(S)\right)\setminus E'_{q+1}\right| + |E^*_q|$$
$$\leq \frac{64d}{\eta}\sum_{S\in\mathcal{S}_q}|E_G(S)| + \frac{64d}{\eta}\cdot|E_G(X_q)|$$
$$\leq \frac{64d}{\eta}\sum_{S\in\mathcal{S}_{q+1}}|E_G(S)|.$$

Therefore, Invariant I4 continues to hold for $\mathcal{S}_{q+1}$. If Invariant I5 continues to hold as well, then we continue to the next iteration. Otherwise, we terminate the first stage, and continue to the second stage.

Consider now the second case, when the algorithm from Lemma 9.2 returns a routing $\mathcal{R}_q$ in G_q of a subset $M^*_q\subseteq M_q$ containing at least $\frac{k_q^{1-22\epsilon}}{2d}$ pairs of vertices, such that every path in $\mathcal{R}_q$ has length at most d, and the total congestion caused by the paths in $\mathcal{R}_q$ is at most η. In this case, we say that iteration q is a *type-2 iteration*. We set $\mathcal{P}_{q+1}=\mathcal{P}_q\cup\mathcal{R}_q$ and $M'_{q+1}=M'_q\cup M^*_q$. Clearly, $\mathcal{P}_{q+1}$ is a routing of the pairs in M'_{q+1} in graph G, and every path in $\mathcal{P}_{q+1}$ has length at most d. Furthermore, since the edges of G that participate in at least $\eta'/2$ paths in $\mathcal{P}_q$ do not lie in graph G_q, and since $\eta<\eta'/2$, we get that the total congestion that the paths of $\mathcal{P}_{q+1}$ cause in graph G is at most η'. We also set $\mathcal{S}_{q+1}=\mathcal{S}_q$. Since $E'_q\subseteq E'_{q+1}$, it is easy to verify that Invariant I4 continues to hold for $\mathcal{S}_{q+1}$, and it is immediate to verify that Invariant I5 holds as well. If $|\mathcal{P}_{q+1}|\geq(1-\alpha)k$, then we terminate the algorithm and return the set $M'=M_{q+1}$ of pairs of vertices and the set $\mathcal{P}=\mathcal{P}_{q+1}$ of paths. From the above discussion, the paths in $\mathcal{P}$ are a routing of the pairs in M'; every path in $\mathcal{P}$ has length at most d, and the paths in $\mathcal{P}$ cause congestion at most η'. Otherwise, if $|\mathcal{P}_{q+1}|<(1-\alpha)k$, then from the above discussion, all invariants hold for $\mathcal{P}_{q+1}$ and $\mathcal{S}_{q+1}$, and we continue to the next iteration.

It remains to consider the case where for every connected component C of graph G_q, the number of demand pairs $(s_i,t_i)\in M_q$ with $s_i,t_i\in V(C)$ is less than $k_q/2$. Let $\mathcal{C}$ denote the set of all connected components of G_q, and let $T'=\{s_i,t_i\mid(s_i,t_i)\in M_q\}$, so $|T'|=2k_q$. For each component $C\in\mathcal{C}$, let $n_C=|V(C)\cap T'|$. Then for all $C\in\mathcal{C}$, $n_C\leq 1.5k_q$ must hold. Let (A',B') be a partition of $V(G_q)$, that is computed as follows. We denote $\mathcal{C}=\{C_1,C_2,\ldots,C_r\}$, where the components are indexed so that $n_{C_1}\geq n_{C_2}\geq\cdots\geq n_{C_r}$. We start with $A'=B'=\emptyset$, and consider the components of $\mathcal{C}$ in the order of their indices. When component C_i is processed, if $|A'\cap T'|\leq|B'\cap T'|$, then we add the vertices of C_i to A', and otherwise we add the vertices of C_i to B'. Consider the partition (A',B') of the vertices of $V(G_q)$ that we obtain at the end of the algorithm, and assume w.l.o.g. that $|A'\cap T'|\geq|B'\cap T'|$. It is easy to verify that $|A'\cap T'|-|B'\cap T'|\leq\max_i\{n_{C_i}\}\leq n_{C_1}\leq 1.5k_q$. Since $|T'|=2k_q$, we then get that $|A'\cap T'|,|B'\cap T'|\geq\frac{k_q}{4}\geq\frac{\alpha k}{8}$. We obtain a cut (A,B) in graph G by letting $A=A'\cup A_q$ and $B=B'$. Clearly, $|A\cap T|,|B\cap T|\geq\frac{\alpha k}{8}$. Next, we show that $|E_G(A,B)|\leq\frac{1024d}{\eta}\cdot\min\{|E_G(A)|,|E_G(B)|\}$. Indeed, it is immediate to verify that $E_G(A,B)\subseteq E'_q$. Since the paths in $\mathcal{P}_q$ have length at most d each, and since $\eta'=\frac{4\eta}{\alpha}$, we get that $|E'_q|\leq\frac{2|\mathcal{P}_q|\cdot d}{\eta'}\leq\frac{\alpha kd}{2\eta}$. On the other hand, since graph G is connected, and since $|A|\geq\frac{\alpha k}{8}$, we get that $|E_G(A)|+|E_G(A,B)|\geq\frac{\alpha k}{16}$. Similarly, $|E_G(B)|+|E_G(A,B)|\geq\frac{\alpha k}{16}$. Altogether, we get that: $|E'_q|\leq\frac{\alpha kd}{2\eta}\leq\frac{8d}{\eta}\cdot\min\{|E_G(A)|,|E_G(B)|\}+\frac{8d}{\eta}|E_G(A,B)|$. Since $\eta\geq 128d$, we get that:

$$|E_G(A,B)|\leq|E'_q|\leq\frac{8d}{\eta}\cdot\min\{|E_G(A)|,|E_G(B)|\}+\frac{|E_G(A,B)|}{2},$$

and so:

$$|E_G(A,B)| \leq \frac{1024d}{\eta} \cdot \min\left\{|E_G(A)|, |E_G(B)|\right\}.$$

In this case, we terminate the algorithm and return the cut (A,B). We say that the current iteration is a type-1 iteration.

This completes the description of the first stage of the algorithm. We now show that the number of iteration in this stage is bounded by q', and bound the running time of the algorithm.

Observation 9.3 *The number of iterations in the algorithm is bounded by $q' = 4d \cdot k^{22\epsilon} \cdot \log k$.*

Proof: We partition the execution of the algorithm into phases. For all $i \geq 1$, the ith phase includes all iterations q, for which $\frac{k}{2^i} < |M_q| \leq \frac{k}{2^{i-1}}$. Clearly, the number of phases is bounded by $\log k$. Next, we bound the number of iterations in a single phase.

Consider some integer i, and denote by $n_i = \frac{k}{2^i}$. If iteration q is a type-2 iteration that belongs to phase i, then $k_q = |M_q| \geq n_i$, and, from Lemma 9.2, at least $\frac{n_i^{1-22\epsilon}}{2d}$ pairs of vertices are routed in iteration q. Therefore, after $2d \cdot n_i^{22\epsilon}$ type-2 iterations, the number of pairs that remain to be routed must decrease by at least factor 2. We conclude that the ith phase may contain at most $2d \cdot n_i^{22\epsilon} \leq 2d \cdot k^{22\epsilon}$ type-2 iterations.

If iteration q is a type-2 iteration in phase i, then we are guaranteed that set X_q contains at least $\frac{n_i^{1-\epsilon}}{32}$ terminals that participate in pairs in M_q. Therefore, after $32n_i^{\epsilon} \leq 32k^{\epsilon}$ type-1 iterations, the number of terminals in $V(G) \setminus A_q$ that remain to be routed (that is, the terminals of M_q), must decrease by at least factor 4. We conclude that a single phase may contain at most $32k^{\epsilon}$ type-1 iterations.

Overall, a single phase may contain at most $2d \cdot k^{22\epsilon} + 32k^{\epsilon} \leq 3d \cdot k^{22\epsilon}$ iterations, and the total number of iterations in the algorithm is bounded by $3dk^{22\epsilon} \log k \leq q'$. □

Since the running time of the algorithm from Lemma 9.2 is bounded by $O\left(m^{1+O(\epsilon)}(\eta + d\log n)\right)$, the total running time of the first stage of the algorithm algorithm is bounded by $O\left(m^{1+O(\epsilon)}(\eta d + d^2 \log n)\right) \leq O\left(m^{1+O(\epsilon)}(\eta d + d^2)\right)$, since $\epsilon > \frac{2}{(\log n)^{1/24}}$, so $n^{\epsilon} > n^{2/(\log n)^{1/24}} > 2^{(\log n)^{3/24}} > \log n$.

Stage 2: Computing the Cut

Assume that the last iteration of the algorithm was iteration q. Let E' be the set of all edges $e \in E(G)$, such that e belongs to at least $\eta'/2$ paths of $\mathcal{P}_q$. Since the paths in $\mathcal{P}_q$ have length at most d each, and since $\eta' = \frac{4\eta}{\alpha}$, we get that $|E'| \leq \frac{2|\mathcal{P}_q| \cdot d}{\eta'} \leq \frac{\alpha k d}{2\eta}$. Let $G' = G \setminus E'$. In the second stage, we will compute a cut (A,B) in graph G', with $|E_{G'}(A,B)| \leq \frac{512d}{\eta} \cdot \min\left\{|E_G(A)|, |E_G(B)|\right\}$, so that each of A, B contains at least $\frac{\alpha k}{8}$ vertices of T. Assume w.l.o.g. that $|E_G(A)| \leq |E_G(B)|$. Since graph G is connected, and since $|A| \geq \frac{\alpha k}{8}$, we get that $|E_G(A)| + |E_G(A,B)| \geq \frac{\alpha k}{16}$. Therefore, $|E'| \leq \frac{\alpha k d}{2\eta} \leq \frac{8d}{\eta}\left(|E_G(A)| + |E_G(A,B)|\right)$. We then get that:

$$\begin{aligned}
|E_G(A,B)| &\leq |E_{G'}(A,B)| + |E'| \\
&\leq \frac{512d}{\eta}|E_G(A)| + \frac{4d}{\eta}\left(|E_G(A)| + |E_G(A,B)|\right) \\
&\leq \frac{516d}{\eta}|E_G(A)| + \frac{4d}{\eta}|E_G(A,B)|.
\end{aligned}$$

Since $\eta \geq 256d$, we get that $|E_G(A,B)| \leq \frac{1024d}{\eta}|E_G(A)| = \frac{1024d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$. In the remainder of the algorithm, it is enough to compute a cut (A,B) in graph G', with $|E_{G'}(A,B)| \leq \frac{512d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$, so that each of A, B contains at least $\frac{\alpha k}{8}$ vertices of T.

Recall that the last iteration of the algorithm was iteration q, and in iteration q we have computed a cut (X_q, Y_q) of the connected component C_q of the corresponding graph G_q. We denote $Y_q' = V(G_q) \setminus X_q$, so $Y_q \subseteq Y_q'$. We have also defined a set $\mathcal{S}_{q+1}$ of disjoint subsets of vertices, with $X_q \in \mathcal{S}_{q+1}$. Let $\mathcal{S}' = \mathcal{S}_{q+1} \cup \{Y_q'\}$. From the description of the algorithm, it is immediate to verify that the subsets of vertices in $\mathcal{S}'$ are all disjoint, and they partition $V(G)$. Since the algorithm terminated at iteration q, we are guaranteed that $|A_{q+1} \cap T| \geq \frac{\alpha k}{2}$. Additionally, if T' denotes the set of all terminals participating in the demand pairs in $\tilde{M}_q$, then at least k_q terminals from T' lie in C_q. Since we have assumed that $|X_q \cap T'| \leq |Y_q \cap T'|$, we get that $|Y_q' \cap T'| \geq \frac{k_q}{2} \geq \frac{\alpha k}{4}$.

Let $E'' = \bigcup_{S \in \mathcal{S}_{q+1}} \delta_{G'}(S) = \left(\bigcup_{S \in \mathcal{S}_{q+1}} \delta_G(S)\right) \setminus E' = \left(\bigcup_{S \in \mathcal{S}_{q+1}} \delta_G(S)\right) \setminus E'_{q+1}$. Recall that we have established that Invariant I4 holds for $\mathcal{S}_{q+1}$, so $|E''| \leq \frac{64d}{\eta} \sum_{S \in \mathcal{S}_{q+1}} |E_G(S)|$.

We now consider two cases. The first case happens if $|E_G(Y_q')| \geq \frac{\eta}{256d}|E''|$. In this case, we consider the cut (A,B) in graph G', where $B = Y_q'$ and $A = V(G) \setminus Y_q'$. From the above discussion $|B \cap T| \geq \frac{\alpha k}{2}$, and $|A \cap T| = |A_{q+1} \cap T| \geq \frac{\alpha k}{2}$. Moreover, $|E_{G'}(A,B)| \leq |E''| \leq \frac{64d}{\eta} \sum_{S \in \mathcal{S}_{q+1}} |E_G(S)| \leq \frac{64d}{\eta}|E_G(A)|$. Altogether, we get that $|E_{G'}(A,B)| \leq |E''| \leq \frac{256d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$, as required.

From now on we consider the second case, where $|E_G(Y_q')| < \frac{\eta}{256d}|E''|$. We compute a partition (A', B') of $V(G) \setminus Y_q'$ as follows. Assume w.l.o.g. that $\mathcal{S}_{q+1} = \{S_1, S_2, \ldots, S_r\}$, where the sets are indexed so that $|E_G(S_1)| \geq |E_G(S_2)| \geq \cdots \geq |E_G(S_r)|$. We start with $A' = B' = \emptyset$, and then consider the sets $S_1, \ldots, S_r$ in this order. When set S_i is considered, if $|E_G(A')| \leq |E_G(B')|$, then we add the vertices of S_i to A', and otherwise we add the vertices of S_i to B'. Consider the partition (A', B') of $V(G) \setminus Y_q'$ that we obtain at the end of this algorithm. If $|A' \cap T| < |B' \cap T|$, then we set $A = A' \cup Y_q'$ and $B = B'$. Otherwise, we set $A = A'$ and $B = B' \cup Y_q'$. We now show that cut (A,B) has all required properties. Assume w.l.o.g. that $|A' \cap T| < |B' \cap T|$ (the other case is symmetric). Then $|B \cap T| = |B' \cap T| \geq \frac{|A_{q+1} \cap T|}{2} \geq \frac{\alpha k}{4}$. Also, $|A \cap T| \geq |Y_q' \cap T| \geq \frac{\alpha k}{2}$. We next show that $|E''| \leq \frac{512d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$ in the following claim.

Claim 9.4

$$|E''| \leq \frac{512d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}.$$

Proof: Recall that we have denoted $\mathcal{S}_{q+1} = \{S_1, \ldots, S_r\}$, where the sets of vertices S_i are indexed in the non-increasing order of the cardinalities of the corresponding sets of edges $E_G(S_i)$. We use the following observation.

Observation 9.5 $\sum_{i=2}^{r} |E_G(S_i)| \geq \frac{\eta}{512d}|E''|$.

We prove Observation 9.5 below, after we complete the proof of Claim 9.4 using it. Since $A' \subseteq A$ and $B' \subseteq B$, it is enough to prove that $|E''| \leq \frac{512d}{\eta} \cdot \min\{|E_G(A')|, |E_G(B')|\}$. Denote $M = \frac{\eta}{512d}|E''|$. We consider two cases. The first case happens if $|E_G(S_1)| \geq M$. Our algorithm then adds the vertices of S_1 to A', and it will keep adding vertices from sets S_i to B' until $|E_G(B')| \geq |E_G(A')| \geq M$ holds. Therefore, we are guaranteed that, at the end of the algorithm, $|E_G(A')|, |E_G(B')| \geq M = \frac{\eta}{512d}|E''|$, and so $|E''| \leq \frac{512d}{\eta} \cdot \min\{|E_G(A')|, |E_G(B')|\}$.

Consider now the second case, where $|E_G(S_1)| < M$. Assume for contradiction that $|E''| > \frac{512d}{\eta} \cdot \min\{|E_G(A')|, |E_G(B')|\}$, and assume w.l.o.g. that $|E_G(A')| \leq |E_G(B')|$, so $|E_G(A')| < \frac{\eta}{512d}|E''|$.

Recall that, from Invariant I4, $\sum_{S\in\mathcal{S}_{q+1}}|E_G(S)|\geq\frac{\eta}{64d}|E''|$. Since $|E_G(A')|<\frac{\eta}{512d}|E''|$, it must be the case that $\sum_{\substack{S\in\mathcal{S}_{q+1}:\\ S\subseteq B'}}|E_G(S)|>\frac{\eta}{128d}|E''|$. Let $S_i\in\mathcal{S}_{q+1}$ be the set whose vertices were added to B' last. Then at the time when the vertices of S_i were added to B', $|E_G(A')|<\frac{\eta}{512d}|E''|$ held. Therefore, from our algorithm, at the same time $|E_G(B')|<\frac{\eta}{512d}|E''|$ held. Therefore, at the end of the algorithm, $\sum_{\substack{S\in\mathcal{S}_{q+1}:\\ S\subseteq B'}}|E_G(S)|\leq\frac{\eta}{512d}|E''|+|E_G(S_i)|$. But since $|E_G(S_i)|\leq|E_G(S_1)|<M=\frac{\eta}{512d}|E''|$, we get that $\sum_{\substack{S\in\mathcal{S}_{q+1}:\\ S\subseteq B'}}|E_G(S)|<\frac{\eta}{256d}|E''|$ holds at the end of the algorithm, a contradiction.

In order to complete the proof of Claim 9.4, it is now enough to prove Observation 9.5.

Proof of Observation 9.5. Let $1\leq i_1<i_2<\cdots<i_r=q$ be the indices of type-2 iterations. Recall that for each $1\leq j\leq r$, in iteration i_j we computed a cut (X_{i_j},Y_{i_j}) of the connected component C_{i_j} of graph G_{i_j}, with $|E_{G_{i_j}}(X_{i_j},Y_{i_j})|\leq\frac{64d}{\eta}\cdot\min\left\{|E_G(X_{i_j})|,|E_G(Y_{i_j})|\right\}$.

We can then denote $\mathcal{S}_{q+1}=\{X_{i_1},X_{i_2},\ldots,X_{i_r}\}$. For all $1\leq j\leq r$, we define a subset $E_j\subseteq E''$ of edges that the set X_{i_j} of vertices is responsible for. We let $E_1=\delta_{G'}(X_{i_1})$, and for $1<j\leq r$, we let $E_j=\delta_{G'}(X_{i_j})\setminus(E_1\cup\cdots\cup E_{j-1})$. It is easy to verify that $(E_1,\ldots,E_r)$ is a partition of E''.

Consider now some index $1\leq j\leq r$, and the cut (X_{i_j},Y_{i_j}) that our algorithm computed in iteration i_j. It is easy to verify that $Y_{i_j}\subseteq X_{i_j+1}\cup\cdots\cup X_{i_r}\cup Y'_q$. Moreover, $E_j=E_{G_{i_j}}(X_{i_j},Y_{i_j})\setminus E'$, and so, from the above discussion:

$$|E_j|\leq\frac{64d}{\eta}\cdot\min\left\{|E_G(X_{i_j})|,|E_G(Y_{i_j})|\right\}.$$

Assume w.l.o.g. that $S_1=X_{i_{j^*}}$. We partition the edges of E'' into three subsets: set $\hat{E}_1=\bigcup_{1\leq j<j^*}E_{i_j}$; set $\hat{E}_2=E_{i_{j^*}}$; and set $\hat{E}_3=\bigcup_{j^*<j\leq r}E_{i_j}$. From the discussion so far, we get that:

$$|\hat{E}_1|\leq\frac{64d}{\eta}\cdot\sum_{1\leq j<j^*}|E_G(X_{i_j})|,$$

$$|\hat{E}_3|\leq\frac{64d}{\eta}\cdot\sum_{j^*<j\leq q}|E_G(X_{i_j})|,$$

and

$$|\hat{E}_2|\leq\frac{64d}{\eta}\cdot|E_G(Y_{i_{j^*}})|.$$

Recall that $Y_{i_{j^*}}\subseteq X_{i_{j^*}+1}\cup\cdots\cup X_{i_r}\cup Y'_q$. Therefore, for every edge $e\in E_G(Y_{i_{j^*}})$, either both endpoints of e lie in one of the sets $X_{i^*_j+1},\cdots,X_{i^*_r},Y'_q$ (in which case $e\in E_G(X_{i_{j^*}+1})\cup\cdots\cup E_G(X_{i_r})\cup E_G(Y'_q)$), or the endpoints of e lie in different sets of $\left\{X_{i_{j^*}+1},\cdots,X_{i_r},Y'_q\right\}$. In the latter case, $e\in E_{i_{j^*}+1}\cup\cdots E_r\cup E'=\hat{E}_3\cup E'$ must hold. Altogether, we get that:

$$\begin{aligned}|E_G(Y_{i_{j^*}})| &\leq \sum_{j^*<j\leq r} |E_G(X_{i_j})| + |E_G(Y'_q)| + |\hat{E}_3| + |E'| \\ &\leq \left(1+\frac{64d}{\eta}\right)\cdot \sum_{j^*<j\leq r} |E_G(X_{i_j})| + |E_G(Y'_q)| + |E'| \\ &\leq \frac{5}{4}\cdot \sum_{j^*<j\leq r} |E_G(X_{i_j})| + |E_G(Y'_q)| + |E'|.\end{aligned}$$

(since $\eta > 256d$).

Recall that $|E'| \leq \frac{\alpha k d}{2\eta} \leq \frac{\alpha k}{512}$, while $|Y_q| \geq |T\cap Y_q| \geq \frac{\alpha k}{4}$. Since Y_q is a set of vertices of a connected component of G_q, $|E_{G_q}(Y_q)| + |E_{G_q}(X_q,Y_q)| \geq \frac{|Y_q|}{2} \geq \frac{\alpha k}{8}$. Moreover, we are guaranteed that $|E_{G_q}(X_q,Y_q)| \leq \frac{64d}{\eta}\cdot |E_{G_q}(Y_q)| \leq \frac{|E_{G_q}(Y_q)|}{4}$. Therefore, $|E_G(Y'_q)| \geq |E_G(Y_q)| \geq |E_{G_q}(Y_q)| \geq \frac{\alpha k}{10}$, and so $|E'| \leq \frac{\alpha k}{512} \leq \frac{|E_G(Y'_q)|}{50}$. Overall, we get that:

$$|E_G(Y_{i_{j^*}})| \leq \frac{5}{4}\cdot \sum_{j^*<j\leq r} |E_G(X_{i_j})| + \frac{51|E_G(Y'_q)|}{50}.$$

Altogether:

$$|\hat{E}_2| \leq \frac{80d}{\eta}\sum_{j^*<j\leq r} |E_G(X_{i_j})| + \frac{80d}{\eta}|E_G(Y'_q)|,$$

and:

$$|E''| = |\hat{E}_1| + |\hat{E}_2| + |\hat{E}_3| \leq \frac{144d}{\eta}\sum_{a=2}^{r}|E_G(S_a)| + \frac{80d}{\eta}|E_G(Y'_q)|.$$

Recall that we have assumed that $|E_G(Y_q)| < \frac{\eta}{256d}|E''|$. Therefore, we get that $|E''| \leq \frac{144d}{\eta}\sum_{a=2}^{r}|E_G(S_a)| + \frac{|E''|}{3}$, and so $\sum_{a=2}^{r}|E_G(S_a)| \geq \frac{\eta}{512d}|E''|$. □ □

Since $E_{G'}(A,B) \subseteq E''$, we get that, in the second case, $|E_{G'}(A,B)| \leq |E''| \leq \frac{512d}{\eta}\cdot\min\{|E_G(A)|, |E_G(B)|\}$ holds. To conclude, we have computed a cut (A,B) in graph G, with $|A\cap T|, |B\cap T| \geq \frac{\alpha k}{8}$ and $|E_G(A,B)| \leq \frac{1024d}{\eta}\cdot\min\{|E_G(A)|, |E_G(B)|\}$.

Recall that the running time of the first stage of the algorithm is at most $O\left(m^{1+O(\epsilon)}(d^2+\eta d)\right)$, while the running time of the second stage can be bounded by $O(m)$. Therefore, the total running time of the algorithm is $O\left(m^{1+O(\epsilon)}(d^2+\eta d)\right)$. In order to prove Theorem 9.1 it now remains to prove Lemma 9.2, which we do next.

9.1 Proof of Lemma 9.2

We denote $T = \{s_1, t_1, \ldots, s_k, t_k\}$, and for convenience we denote by $\tilde{k} = |T| = 2k$.

The algorithm consists of two stages. In the first stage, we apply the algorithm from Corollary 5.3 to graph G and the set T of terminals. If the algorithm returns two subsets $T_1, T_2 \subseteq T$ of terminals

and a set E' of edges (essentially defining a distancing (T_1, T_2, E')), then we will use the algorithm from Lemma 4.1 in order to convert this distancing into a low-conductance cut (A, B) as required. Otherwise, the algorithm must embed a large graph H into G, and construct a level-$(1/\epsilon)$ hierarchical support structure for H, so that H is $(\eta', \tilde{d})$-well-linked with respect to the set $S(H)$ of vertices defined by the hierarchical support structure. In this latter case, we continue to Stage 2. In Stage 2, we will initialize an ES-Tree data structure, rooted at the set $S(H)$ of vertices in graph G. Additionally, we will maintain a data structure from Theorem 7.1 in order to support approximate short-path queries in graph H, as it undergoes edge deletions. We will use these data structure to iteratively identify pairs $(s_i, t_i) \in M$ of vertices, that can be connected via a short path P_i in G. The corresponding path P_i is then added to the routing $\mathcal{P}$ that we are constructing, and every edge of G that currently appears on η paths in $\mathcal{P}$ is deleted from G. Edges of H whose embedding paths have thus been eliminated will be deleted from H. The algorithm terminates when we can no longer route the remaining paths of $\mathcal{P}$ via short paths. If, by that time $|\mathcal{P}| \geq z$, then we return the routing in $\mathcal{P}$. Otherwise, we will obtain a distancing in graph G, that can again be converted into a low-conductance cut. Before we describe each of the two stages, we need to consider an easier special case where $k \leq n^{\epsilon}$.

9.1.1 Special Case: $k \leq n^{\epsilon}$

For all $1 \leq i \leq k$, let $A_i = \{s_i\}$ and $B_i = \{t_i\}$. We apply Procedure ProcPathPeel from Lemma 3.12 to graph G and the collections $A_1, B_1, \ldots, A_k, B_k$ of subsets of its vertices, and parameters d and η. Let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ be the collection of paths that the algorithm outputs. Then for all $1 \leq i \leq k$, either $\mathcal{P}_i = \emptyset$, or $\mathcal{P}_i$ contains a single path P_i, connecting s_i to t_i in G. Let $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}_i$, and let $M' \subseteq M$ contain all pairs $(s_i, t_i) \in M$, such that some path in $\mathcal{P}$ connects s_i to t_i. Clearly, $\mathcal{P}$ is a routing of the pairs in M', and we are guaranteed that every path in $\mathcal{P}$ has length at most d, and the paths in $\mathcal{P}$ cause congestion at most η. Let E' be the set of all edges of G that participate in exactly η paths in $\mathcal{P}$, and let $M'' = M \setminus M'$. From Property P3, for every pair $(s_i, t_i) \in M''$ of vertices, $\mathsf{dist}_{G \setminus E'}(s_i, t_i) > d$. Notice also that $|E'| \leq \frac{\sum_{P \in \mathcal{P}} |E(P)|}{\eta} \leq \frac{d \cdot |\mathcal{P}|}{\eta}$. Recall that the running time of the algorithm is $O(m\eta + mdk \log n) \leq O(mn^{O(\epsilon)}d + m\eta) \leq O\left(m^{1+O(\epsilon)}(\eta + d \log n)\right)$.

We now consider two cases. The first case happens if $|\mathcal{P}| \geq z$. In this case, we return the routing $\mathcal{P}$ of the set M' of pairs of vertices.

Consider now the second case, where $|\mathcal{P}| < z$. In this case, $|M''| \geq k - z \geq k/2$. Let $T' \subseteq T$ be the set of all vertices that participate in the pairs in M'', and let $\tilde{G} = G \setminus E'$. Since, for every pair $(s_i, t_i) \in M''$ of vertices, $\mathsf{dist}_{\tilde{G}}(s_i, t_i) > d$, we get that for every vertex $x \in T'$, $|B_{\tilde{G}}(x, d/2) \cap T'| < |T'|/2$. Let $\Delta = \frac{64}{\epsilon}$ and $\hat{d} = \frac{d}{2\Delta} = \frac{d\epsilon}{128}$. We apply Procedure ProcSeparate from Lemma 3.10 to graph $\tilde{G}$, the set T' of terminals, parameters Δ, $\alpha = 1/3$, and distance parameter $\hat{d}$ replacing d. Note that the algorithm may not return a terminal $t \in T'$ with $|B_{\tilde{G}}(t, \Delta \cdot \hat{d}) \cap T'| = |B_{\tilde{G}}(t, d/2) \cap T'| > \alpha |T'|$, since, as observed above, for each such terminal t, $|B_{\tilde{G}}(t, d/2) \cap T'| < |T'|/2$.

Therefore, the algorithm must return two subsets T_1, T_2 of terminals, with $|T_1| = |T_2|$, such that $|T_1| \geq \frac{|T'|^{1-64/\Delta}}{3} \geq \frac{k^{1-\epsilon}}{16}$. Moreover, for every pair $t \in T_1$, $t' \in T_2$ of terminals, $\mathsf{dist}_{\tilde{G}}(t, t') \geq \hat{d} = \frac{d\epsilon}{128}$. Recall that the running time of Procedure ProcSeparate is bounded by $O(m \cdot |V(G)|^{64/\Delta}) \leq O(m^{1+O(\epsilon)})$.

Recall that $|E'| \leq \frac{d|\mathcal{P}|}{\eta} \leq \frac{dz}{\eta} = \frac{k^{1-22\epsilon}}{\eta} \leq \frac{16d|T_1|}{\eta}$. Clearly, (T_1, T_2, E') is a $(\delta, \hat{d})$-distancing in graph G, for some parameter $0 < \delta < 1$. Let $\varphi = \frac{64d}{\eta}$, so that $|E'| \leq \frac{\varphi |T_1|}{4}$. Since $\eta > 128d$, we get that $\varphi < 1/2$. Notice also that:

$$\hat{d} = \frac{d\epsilon}{128} \geq \frac{\eta \log m}{d} = \frac{32 \log m}{\varphi},$$

since $\eta \leq \frac{d^2}{2^{c^*/\epsilon^6} \cdot \log m} \leq \frac{d^2\epsilon}{128 \log m}$ from the statement of Lemma 9.2.

We can now use the algorithm from Lemma 4.1 to compute a cut (A, B) in graph G, with $T_1 \subseteq A$, $T_2 \subseteq B$, such that $|E_G(A,B)| \leq \varphi \cdot \min\{|E_G(A)|, |E_G(B)|\} \leq \frac{64d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$. We return the cut (A, B) as the outcome of the algorithm. From the above discussion, $|A \cap T|, |B \cap T| \geq |T_1| \geq \frac{k^{1-\epsilon}}{16}$.

The running time of the algorithm from Lemma 4.1 is bounded by $O(m)$, and so the total running time in Case 1 is bounded by $O\left(m^{1+O(\epsilon)}(\eta + d \log n)\right)$.

From now on, we assume that $k \geq n^{\epsilon}$. The remainder of the algorithm consists of two stages, that we now describe.

9.1.2 Stage 1: Embedding a Well-Connected Graph

In this stage, we will apply the algorithm from Corollary 5.3 to graph G and the set T of terminals. In order to be able to do so, we need to ensure that $\epsilon > \frac{2}{(\log \tilde{k})^{1/12}}$ holds. Recall that $\tilde{k} = 2k \geq 2n^{\epsilon}$ from our assumption. Therefore:

$$\frac{2^{12}}{\log \tilde{k}} \leq \frac{2^{12}}{\log(2n^{\epsilon})} \leq \frac{2^{12}}{\epsilon \cdot \log n}.$$

From our assumption that $\epsilon > \frac{2}{(\log n)^{1/24}}$, we get that $\log n > \frac{2^{24}}{\epsilon^{24}}$. Therefore:

$$\frac{2^{12}}{\log \tilde{k}} \leq \frac{\epsilon^{23}}{2^{12}}.$$

We conclude that:

$$\epsilon > \frac{2}{(\log \tilde{k})^{1/23}} \geq \frac{2}{(\log \tilde{k})^{1/12}}. \tag{13}$$

Let $d' = \frac{8d}{2^{c^*/\epsilon^6}}$ and $\eta' = \frac{8\eta}{2^{c^*/\epsilon^6}}$. We apply the algorithm from Corollary 5.3 to graph G, set T of terminals, parameters d', η', and parameter ϵ that remains unchanged. Recall that the running time of the algorithm is:

$$O\left(k^{1+O(\epsilon)} + m \cdot k^{O(\epsilon^3)} \cdot (\eta' + d' \log m)\right) \leq O\left(m^{1+O(\epsilon)}(d' + \eta')\right) \leq O\left(m^{1+O(\epsilon)}(d + \eta)\right).$$

We now consider two cases.

Case 1. The first case happens if the algorithm from Corollary 5.3 returns a pair $T_1, T_2 \subseteq T$ of disjoint subsets of terminals, and a set E' of edges of G, such that $|T_1| = |T_2|$ and $|T_1| \geq \frac{\tilde{k}^{1-4\epsilon^3}}{4} \geq \frac{k^{1-4\epsilon^3}}{4}$. Recall that the algorithm also guarantees for every pair $t \in T_1, t' \in T_2$ of terminals, $\mathsf{dist}_{G \setminus E'}(t, t') > d'$.

Moreover, we are guaranteed that: $|E'| \leq \frac{d' \cdot |T_1|}{\eta'} = \frac{d \cdot |T_1|}{\eta}$. Denote $\varphi = \frac{4d}{\eta}$, so that $|E'| \leq \frac{\varphi |T_1|}{4}$ holds. Since $\eta > 128d$, we get that $\varphi < 1/2$. Since $\eta \leq \frac{d^2}{2^{c^*/\epsilon^6} \cdot \log m}$, we get that:

$$\frac{32 \log m}{\varphi} = \frac{8\eta \log m}{d} \leq \frac{8d}{2^{c^*/\epsilon^6}} = d'.$$

We can now apply the algorithm from Lemma 4.1, to compute a cut (A,B) in graph G, with $T_1 \subseteq A$, $T_2 \subseteq B$, such that $|E_G(A,B)| \leq \varphi \cdot \min\{|E_G(A)|, |E_G(B)|\} = \frac{4d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$. The running time of this algorithm is bounded by $O(m)$. We return the cut (A,B) as the output of the algorithm. Clearly, $|A \cap T|, |B \cap T| \geq |T_1| \geq \frac{k^{1-4\epsilon^3}}{4} \geq \frac{k^{1-\epsilon}}{16}$.

Case 2. In the second case, the algorithm from Corollary 5.3 must return a graph H with $V(H) \subseteq T$, $|V(H)| = N^{1/\epsilon} \geq \tilde{k} - \tilde{k}^{1-\epsilon/2}$, where $N = \left\lfloor \tilde{k}^{\epsilon} \right\rfloor = \lfloor (2k)^{\epsilon} \rfloor$, so that the maximum vertex degree in H is at most $\tilde{k}^{32\epsilon^3}$.

Recall that the algorithm must also return an embedding $\mathcal{P}^*$ of H into G via paths of length at most d', that cause congestion at most $\eta' \cdot \tilde{k}^{32\epsilon^3}$, and a level-$(1/\epsilon)$ hierarchical support structure for H, such that H is $(\tilde{\eta}, \tilde{d})$-well-connected with respect to the set $S(H)$ of vertices defined by the support structure, where $\tilde{\eta} = N^{6+256\epsilon}$, and $\tilde{d} = 2^{c/\epsilon^5}$, with c being the constant used in the definition of the Hierarchical Support Structure. In this case, we continue to Stage 2 of the algorithm.

This completes the description of the first stage of the algorithm. From the above analysis, the running time of this stage is bounded by $O\left(m \cdot n^{O(\epsilon)}(d+\eta)\right)$.

9.1.3 Stage 2: Computing the Routing

In this stage, we start with $\mathcal{P} = \emptyset$, and then gradually add paths to $\mathcal{P}$. We also maintain a graph $\tilde{G}$, where initially $\tilde{G} = G$. As the algorithm progresses, every edge $e \in E(\tilde{G})$ that participates in η paths of $\mathcal{P}$ is deleted from $\tilde{G}$. Therefore, we can think of graph $\tilde{G}$ as a dynamic graph, with edges deleted from $\tilde{G}$ over time. Whenever an edge e is deleted from graph $\tilde{G}$, for every edge $e' \in E(H)$ whose embedding path $P(e') \in \mathcal{P}^*$ contains e, we also delete edge e' from graph H. Therefore, graph H can also be viewed as a dynamic graph. We will maintain the following data structures throughout the algorithm.

The first data structure, that we denote by $\mathcal{D}(H)$, will be used in order to support approximate shortest-path queries in graph H. The data structure is maintained using the algorithm from Theorem 7.1. Recall that $N = \left\lfloor \tilde{k}^{\epsilon} \right\rfloor$, and that we have established in Inequality 13 that $\frac{2}{(\log \tilde{k})^{1/12}} < \epsilon < 1/400$. In order to be able to use Theorem 7.1, we need to verify that $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^6}$ holds. Indeed observe first that, since $\epsilon > \frac{2}{(\log \tilde{k})^{1/12}}$, we get that

$$\tilde{k}^{\epsilon^8} > \tilde{k}^{(2/(\log \tilde{k})^{1/12})^8} = \tilde{k}^{256/(\log \tilde{k})^{2/3}} > 2^{(\log \tilde{k})^{1/3}} > \log \tilde{k}.$$

Additionally, from the inequality $\epsilon > \frac{2}{(\log \tilde{k})^{1/12}}$, we get that $\log \tilde{k} \geq (2/\epsilon)^{12}$, and:

$$\tilde{k} \geq 2^{(2/\epsilon)^{12}} \tag{14}$$

Lastly, since $N = \left\lfloor \tilde{k}^{\epsilon} \right\rfloor$ and $\epsilon < 1/400$, we get that:

$$\frac{N^{\epsilon^4}}{\log N} \geq \frac{\tilde{k}^{\epsilon^6}}{\epsilon \cdot \log \tilde{k}} \geq \tilde{k}^{\epsilon^6/2} \geq 2^{1/\epsilon^6} \geq 2^{128/\epsilon^6}.$$

We let $q = 1/\epsilon$. We maintain the data structure from Theorem 7.1 in graph H, with parameters $j = q$, and parameters N and ϵ that remain unchanged. Recall that we are given a level-q hierarchical support

structure for H, such that H is $(\tilde{\eta}, \tilde{d})$-well-connected with respect to the set $S(H)$ of vertices defined by the support structure, where $\tilde{\eta} = N^{6+256\epsilon} = N^{6+256q\epsilon^2} = \eta_q$, and $\tilde{d} = 2^{c/\epsilon^5} = 2^{cq/\epsilon^4} = \tilde{d}_q$, where η_q and $\tilde{d}_q$ are the parameters that are used in the definition of the Hierarchical Support Structure.

We denote the data structure that the algorithm from Theorem 7.1 maintains by $\mathcal{D}(H)$. Recall that this data structure can withstand the deletion of up to $\Lambda_q = N^{q-8-300q\epsilon^2} = N^{q-8-300\epsilon}$ edge deletions from graph H. As long as fewer than Λ_q edges are deleted from H, the algorithm maintains a decremental set $S'(H) \subseteq V(H)$ of vertices, with $|S'(H)| \geq \frac{N^q}{16^q} = \frac{|V(H)|}{16^q} \geq \frac{k}{2^{8/\epsilon}}$.

The algorithm supports short-path queries between vertices of $S'(H)$: given a pair $x, y \in S'(H)$ of vertices, return a path P connecting x to y in the current graph H, whose length is at most $d^*_q = 2^{O(q/\epsilon^5)} = 2^{O(1/\epsilon^6)}$, in time $O(|E(P)|)$.

If we denote by m' the number of edges in H at the beginning of the algorithm, then the total update time of the algorithm is bounded by:

$$O\left(qN^{q+3} \cdot 2^{O(1/\epsilon^6)} + m' \cdot N^2 \cdot 2^{O(1/\epsilon^6)}\right).$$

Recall that $N \leq \tilde{k}^\epsilon \leq n^\epsilon$, and maximum vertex degree in H is at most $n^{O(\epsilon^3)}$. Recall also that $N^q \leq n$, and $q = 1/\epsilon$. Furthermore, as established in Inequality 14, $n \geq \tilde{k} \geq 2^{(2/\epsilon)^{12}}$, and so $2^{O(1/\epsilon^6)} \leq n^{O(\epsilon)}$. It is then easy to verify that the running time of the algorithm is bounded by $O\left(n^{1+O(\epsilon)}\right)$.

The second data structure is, intuitively, an ES-Tree in G that is rooted at the set $S'(H)$ of vertices, and whose depth parameter is $d/2$. Specifically, we maintain a graph G', that is defined as follows. Initially, we obtain graph G' from G, by adding a source vertex s, that connects to every vertex $v \in S(H)$ with an edge. As the algorithm progresses and new paths are added to the set $\mathcal{P}$ of paths that we construct, whenever some edge $e \in E(G)$ appears in η paths of $\mathcal{P}$, we delete e from G'. For every edge $e' \in E(H)$, whose embedding path $P(e') \in \mathcal{P}^*$ contains edge e, we also delete e' from H, and update data structure $\mathcal{D}(H)$ accordingly. If, as the result of this update, some vertex x is deleted from set $S'(H)$, then we also delete edge (s, x) from graph G'. We maintain an ES-Tree data structure in graph G', rooted at vertex s, with depth bound $d/2 + 1$. We denote the data structure, and the corresponding tree, by τ. The total update time that is needed in order to maintain the ES-Tree data structure τ is bounded by $O(md \log n)$.

Lastly, for every edge $e \in E(G)$, we maintain a list $L(e)$ of all edges $e' \in E(H)$, such that the embedding path $P(e') \in \mathcal{P}^*$ contains e. We also maintain a pointer from e to every edge in $L(e)$ and back. Recall that the embedding $\mathcal{P}^*$ causes congestion at most $\eta' \cdot \tilde{k}^{32\epsilon^3}$, so the length of each such list is bounded by $\eta' \cdot \tilde{k}^{32\epsilon^3}$. Moreover, the deletion of an edge $e \in E(G)$ from graph G' may trigger the deletion of at most $\eta' \cdot \tilde{k}^{32\epsilon^3}$ edges from graph H. We will ensure that $|\mathcal{P}| \leq z$ holds throughout the algorithm, and that every path in $\mathcal{P}$ has length at most d. Therefore, if we denote by $E' \subseteq E(G)$ the collection of all edges that participate in η paths in $\mathcal{P}$, the we are guaranteed that throughout the algorithm:

$$|E'| \leq \frac{\sum_{P \in \mathcal{P}} |E(P)|}{\eta} \leq \frac{|\mathcal{P}| \cdot d}{\eta} \leq \frac{zd}{\eta}.$$

Therefore, the total number of edges that may be deleted from graph H over the course of the algorithm is bounded by:

$$\begin{aligned}\frac{zd}{\eta}\cdot\eta'\cdot\tilde{k}^{32\epsilon^3} &\leq \tilde{k}^{1-22\epsilon}\cdot\tilde{k}^{32\epsilon^3}\\ &\leq \tilde{k}^{1-21\epsilon}\\ &\leq \frac{N^q\cdot 2^q}{N^{21\epsilon q}}\\ &< N^{q-9} < \Lambda_q\end{aligned}$$

(For the first inequality, we have used the fact that $z=\frac{k^{1-22\epsilon}}{d}$, $k\leq\tilde{k}$, and $\eta'=\frac{8\eta}{2^{c^*/\epsilon^6}}\leq\eta$. For the third inequality we used the fact that $N=\left\lfloor\tilde{k}^{\epsilon}\right\rfloor$, and $q=1/\epsilon$, so $N^q\leq\tilde{k}\leq N^q\cdot 2^q$ holds. The fourth inequality follows since $2^q=2^{1/\epsilon}<\left(\frac{\tilde{k}^{\epsilon}}{2}\right)^9\leq N^9$ from Inequality 14.)

If (s_i,t_i) is a pair of vertices in M, then we say that s_i is a *mate* of t_i, and t_i is a mate of s_i. Throughout the algorithm, we will also maintain a subset $S''(H)\subseteq S'(H)$ containing all vertices $x\in S'(H)$, such that no path in $\mathcal{P}$ has x as its endpoint, and the mate of x still lies in the tree τ. We also maintain, for every edge $e\in E(G)$, a counter $n(e)$, counting the number of paths in $\mathcal{P}$ that contain e.

We are now ready to describe our algorithm. The algorithm consists of at most z iterations, and they are performed as long as $|\mathcal{P}|<z$ and $S''(H)\neq\emptyset$ hold.

In order to perform a single iteration, we let x be any vertex in set $S''(H)$. Assume w.l.o.g. that $x=s_i$, and that its mate is t_i. Since $x\in S''(H)$, vertex t_i currently lies in the tree τ. Using the tree, we can compute a path Q in graph G', that connects t_i to some vertex $y\in S(H')$, so that the length of the path is bounded by $d/2$, and the path does not contain any vertices of $S'(H)\setminus\{y\}$. This can be done in time $O(|E(Q)|)$. Next, using data structure $\mathcal{D}(H)$, we compute a path $\tilde{Q}'$ connecting y to s_i in graph H, such that the length of the path is bounded by $2^{O(1/\epsilon^6)}$. This can be done in time $O(|E(\tilde{Q}')|)$. Lastly, by replacing every edge e' on path $\tilde{Q}'$ with its embedding path $P(e')\in\mathcal{P}^*$, we obtain a path Q' in graph G, connecting y to s_i, whose length is bounded by:

$$2^{O(1/\epsilon^6)}\cdot d'\leq d/2,$$

since $d'=\frac{8d}{2^{c^*/\epsilon^6}}$, and we can assume that c^* is a large enough constant. By combining the paths Q and Q', we obtain a path P, connecting s_i to t_i in graph G, whose length is at most d. If path P is not simple, then we convert it into a simple path, in time $O(d)$. This can be done, for example, by traversing the vertices of P in the order of their appearance on the path, and marking every vertex that has been traversed in an array of length n. Whenever we attempt to mark a vertex v that was already marked before, we recognize that we closed a simple cycle $C\subseteq P$. We can then retrace this cycle and un-mark all its vertices except for v. We delete all vertices of $C\setminus\{v\}$ from P, and continue the traversal of P starting from v. Since every vertex on P may be traversed at most twice (once when we visit it for the first time, and the second time when we remove a cycle on which that vertex lies), this algorithm takes time $O(d)$, assuming that we are provided an empty array of length n, that can be used in order to mark and un-mark the vertices of P. At the end of this procedure, we un-mark every vertex of P, so the array can be reused in the next iteration. We denote the resulting simple path P by P_i, and we add P_i to the set $\mathcal{P}$ of paths that we maintain.

Next, we consider every edge $e\in E(P_i)$ one by one. For each such edge e, we increase the counter $n(e)$ by 1. If $n(e)=\eta$ holds, then we delete e from graph G', and update data structure τ accordingly. Additionally, fo every edge $e'\in L(e)$, we delete e' from graph H, updating the data structure $\mathcal{D}(H)$

accordingly, and we update all lists $L(e'')$ of edges $e'' \in E(G)$ with $e' \in L(e'')$, by deleting e' from $L(e'')$.

As the result of these updates to data structure $\mathcal{D}(H)$, we may have deleted some vertices from set $S'(H)$. Let Y be the set of all such vertices. For every vertex $v \in S'(H)$, we delete the edge (s, v) from graph G', and update the data structure τ accordingly. We also delete v from set $S''(H)$ if it belongs to this set.

Lastly, whenever a vertex u leaves tree τ, if either u or its mate u' lie in $S''(H)$, then we delete both vertices from $S''(H)$. We also delete s_i and t_i from $S''(H)$. This completes the description of an iteration. Besides the time that is needed in order to maintain data structures $\mathcal{D}(H), \tau, S''(H)$, and $\{L(e), n(e)\}_{e \in E(G)}$, the additional time that is needed in order to execute an iteration is bounded by $O(d)$.

We now consider two cases. In the first case, the algorithm terminates with $|\mathcal{P}| \geq z$. In this case, we return the set $\mathcal{P}$ of paths as the algorithm's outcome, together with the collection $M' \subseteq M$ of pairs of vertices, containing every pair $(s_i, t_i) \in M$ for which some path connecting s_i to t_i lies in $\mathcal{P}$. It is easy to verify that the set $\mathcal{P}$ of paths has all required properties.

Consider now the second case, when $|\mathcal{P}| < z = \frac{k^{1-22\epsilon}}{d} < \frac{k}{2^{16/\epsilon}}$ (from Inequality 14). In this case, the algorithm must have terminated because $S''(H) = \emptyset$ holds. Since the number of vertices that serve as endpoints of paths in $\mathcal{P}$ is bounded by $2z < \frac{2k}{2^{16/\epsilon}}$, while we are guaranteed that $|S'(H)| \geq \frac{k}{2^{8/\epsilon}}$, there must be a set $X \subseteq S'(H)$ of at least $\frac{|S'(H)|}{2} \geq \frac{k}{2^{16/\epsilon}}$ vertices, such that, for every vertex $x \in X$, no path in $\mathcal{P}$ has x as its endpoint, and the mate of x does not belong to the tree τ. Therefore, if we denote by X' the set of vertices that are mates of the vertices of X, then, in the current graph G', $\mathsf{dist}_{G'}(X, X') > d/2$. Clearly, $X' \cap X = \emptyset$, and $|X'| = |X|$. Let E' be the set of all edges of graph G that belong to η paths of $\mathcal{P}$. Then $|E'| \leq \frac{|\mathcal{P}| \cdot d}{\eta} \leq \frac{d}{\eta} \cdot |X|$. Moreover, $\mathsf{dist}_{G \setminus E'}(X, X') > d/2$. Therefore, (X, X', E') is a $(\delta, d/2)$-distancing in graph G, for some parameter $0 < \delta < 1$.

We denote $\varphi = \frac{4d}{\eta}$, so that $|E'| \leq \frac{\varphi |X|}{4}$ holds. Notice that: $\frac{32 \log m}{\varphi} = \frac{8\eta \log m}{d} < \frac{d}{2}$, since $\eta \leq \frac{d^2}{2^{c^*/\epsilon^6} \cdot \log m}$ from the statement of Lemma 9.2. We can now apply the algorithm from Lemma 4.1, to compute a cut (A, B) in graph G, with $X \subseteq A$, $X' \subseteq B$, and $|E_G(A, B)| \leq \varphi \cdot \min\{|E_G(A)|, |E_G(B)|\} = \frac{4d}{\eta} \cdot \min\{|E_G(A)|, |E_G(B)|\}$. Recall that $|X|, |X'| \geq \frac{k}{2^{16/\epsilon}} \geq \frac{k^{1-\epsilon}}{16}$, since $2^{16/\epsilon} \leq 16k^{\epsilon}$ from Inequality 14. We then return the cut (A, B) as the output of the algorithm. The running time of the algorithm from Lemma 4.1 is $O(m)$.

It now remains to analyze the running time of the algorithm. The running time of Stage 1 is bounded by $O\left(m^{1+O(\epsilon)}(d + \eta)\right)$, as shown already.

In order to analyze the running time of Stage 2, recall that the total update time for maintaining data structure $\mathcal{D}(H)$, as established already, is bounded by $O(n^{1+O(\epsilon)})$, and the total update time for maintaining data structure τ is bounded by $O(md \log n)$. Since the paths in $\mathcal{P}^*$ cause congestion at most $\eta' \cdot k^{O(\epsilon^3)}$, maintaining the lists $\{L(e)\}_{e \in E(G)}$ takes total time $O(m \cdot \eta' \cdot k^{O(\epsilon^3)}) \leq O(m\eta k^{O(\epsilon^3)})$. The time required for additional computation in every iteration (that is, computing the path P, converting it into a simple path, and reducing the counters $n(e)$ for all edges $e \in E(P)$) is bounded by $O(d)$. The number of iterations is bounded by $O(k)$. The running time required for the remaining calculations that the algorithm performs (such as, for example, applying the algorithm from Lemma 4.1 to compute a low-conductance cut at the end of the algorithm if $|\mathcal{P}| < z$) is subsumed by the above running times. The total running time is then bounded by:

$$O\left(m^{1+O(\epsilon)}(d + \eta)\right) + O(n^{1+O(\epsilon)}) + O(md \log n) + O(m \cdot \eta \cdot k^{O(\epsilon^3)}) \leq O\left(m^{1+O(\epsilon)}(d + \eta)\right).$$

10 An Algorithm for the Cut Player in the Cut-Matching Game – Proof of Theorem 2.6

In this section we provide an algorithm for the Cut Player in the Cut-Matching Game, proving Theorem 2.6. The algorithm consists of a number of phases. At the beginning of phase q, we are given a partition (X_q, Y_q) of $V(G)$, with $|X_q| \geq \frac{3n}{4}$, and $|E_G(X_q, Y_q)| \leq \frac{|Y_q|}{100}$. At the beginning of the algorithm, we use the partition (X_1, Y_1) of $V(G)$, with $X_1 = V(G)$ and $Y_1 = \emptyset$. We now describe the execution of Phase q.

Let $G_q = G[X_q]$. Assume first that, for every connected component C of graph G_q, $|V(C)| \leq 5n/8$. Let C be the largest connected component of G_q. If $|V(C)| \geq n/4$, then we return a cut (A, B) in graph G with $A = V(C)$ and $B = V(G) \setminus V(C)$. Clearly, $|A| \geq n/4$, and, since $|A| \leq 5n/8$, we get that $|B| \geq n/4$ as well. Moreover, $E_G(A, B) \subseteq E_G(X_q, Y_q)$, and so $|E_G(A, B)| \leq \frac{|Y_q|}{100} \leq \frac{n}{100}$. Otherwise, if $|V(C)| < n/4$, then we compute a partition (A, B) of $V(G)$ as follows. We start with $A = \emptyset$ and $B = Y_q$, and then process every connected component C' of G_q one by one. When a component C' is processed, if $|A| \leq |B|$ holds, then we add the vertices of C' to A, and otherwise we add them to B. Since every connected component of G_q contains at most $n/4$ vertices, and $|Y_q| \leq n/4$, it is easy to verify that at the end of the algorithm, $|A|, |B| \geq n/4$ holds. As before, $E_G(A, B) \subseteq E_G(X_q, Y_q)$, and so $|E_G(A, B)| \leq \frac{|Y_q|}{100} \leq \frac{n}{100}$. We terminate the algorithm and return the cut (A, B).

We assume from now on that there is a connected component G'_q of graph G_q with $|V(G'_q)| \geq 5n/8$. We denote $n_q = |V(G'_q)|$. We use algorithm ConstructExpander from Theorem 3.3 to construct a graph H_q, with $|V(H_q)| = n_q$, such that H_q is an α_0-expander, and maximum vertex degree in H_q is at most 9. The running time of this algorithm is $O(n_q) \leq O(n)$. It will be convenient for us to identify the vertices of H_q with the vertices of G'_q. In other words, we assume that $V(H_q) = V(G'_q)$, by arbitrarily mapping every vertex of H_q to a distinct vertex of G'_q.

Using a simple standard greedy algorithm, we compute, in time $O(n)$, a partition $M_1, M_2, \ldots, M_{19}$ of the set $E(H_q)$ of edges, so that, for each $1 \leq i \leq 19$, M_i is a matching. We use a parameter $\rho = \frac{n}{2^{\hat{c}/\epsilon^6} \cdot \Delta^3 \cdot \log^2 n}$, where $\hat{c}$ is a large enough constant.

Next, we perform 19 iterations. For $1 \leq i \leq 19$, in the ith iteration, we use the algorithm for advanced path peeling from Theorem 2.5 in order to embed the edges of M_i into graph G'_q with low congestion. If we successfully embed all but at most ρ edges of M_i, then we partition the set M_i of edges into two subsets: set M'_i containing all edges that we managed to embed, and set F_i containing all remaining edges. We say that the ith iteration ended with a routing, and continue to the next iteration. If we failed to embed a large enough subset of M_i into G'_q, then we will compute a sparse cut in graph G'_q, that will allow us to update the partition (A_q, B_q) of $V(G)$. We then say that the ith iteration ended with a cut, and terminate the current phase. If every one of the 19 iterations ended with a routing, then, by letting letting G''_q be the graph that is obtained from G'_q by adding to it a set $\bigcup_{i=1}^{19} F_i$ of fake edges, we obtain an embedding of H_q into G''_q with low congestion. We can then use Algorithm AlgExtractExpander from Lemma 3.5, to extract a large enough expander graph $G^* \subseteq G'_q$. We now describe the execution of a single iteration.

Consider an index $1 \leq i \leq 19$. If $|M_i| \leq \rho$, then we let $M'_i = \emptyset$, $\mathcal{P}_i = \emptyset$, $F_i = M_i$, and continue to the next iteration. From now on we assume that $|M_i| > \rho$.

We apply the algorithm from Theorem 2.5 to graph G'_q and the set M_i of pairs of its vertices, with parameters ϵ, $\varphi = \frac{1}{100\Delta}$, and $\alpha = \frac{\rho}{2|M_i|}$, so $0 < \alpha \leq \frac{1}{2}$ holds. Recall that $\frac{2}{(\log n)^{1/25}} < \epsilon < \frac{1}{400}$, and so $\log n > \left(\frac{2}{\epsilon}\right)^{25}$. Since $n_q > n/2$, we get that $\log(n_q) \geq \log n - 1 \geq \left(\frac{2}{\epsilon}\right)^{24}$. Therefore, the condition of Theorem 2.5 that $\frac{2}{(\log n)^{1/24}} < \epsilon < \frac{1}{400}$ holds. We denote by T_i the set of all vertices of G'_q that serve

as endpoints of the edges of M_i, so $|T_i| = 2|M_i| \geq 2\rho$.

We now consider two cases. In the first case, the algorithm from Theorem 2.5 returns a cut (A_q, B_q) in G'_q, with $|E_{G'_q}(A_q, B_q)| \leq \varphi \cdot \min\{|E_G(A_q)|, |E_G(B_q)|\}$, such that $|A_q \cap T_i|, |B_q \cap T_i| \geq \frac{\alpha \cdot |M_i|}{16} \geq \frac{\rho}{32}$. In this case we say that iteration i ended with a cut. Since maximum vertex degree in G is at most Δ, we get that $|E_G(A_q)| \leq \Delta \cdot |A_q|$ and $|E_G(B_q)| \leq \Delta \cdot |B_q|$, and since $\varphi = \frac{1}{100\Delta}$, we get that $|E_{G'_q}(A_q, B_q)| \leq \frac{1}{100\Delta} \min\{|E_G(A_q)|, |E_G(B_q)|\} \leq \frac{1}{100} \cdot \min\{|A_q|, |B_q|\}$. Assume w.l.o.g. that $|A_q| \geq |B_q|$. Since we have assumed that $|V(G'_q)| \geq \frac{5n}{8}$, we get that $|A_q| \geq \frac{5n}{16} \geq \frac{n}{4}$. We set $X_{q+1} = A_q$ and $Y_{q+1} = V(G) \setminus A_q$. Clearly, (X_{q+1}, Y_{q+1}) is a partition of $V(G)$, and, from our discussion, $|X_{q+1}| \geq \frac{n}{4}$. Moreover, we can think of the cut (X_{q+1}, Y_{q+1}) as obtained from cut (X_q, Y_q), by moving all vertices of $V(G_q) \setminus A_q$ from X_q to Y_q. Therefore, $|E_G(X_{q+1}, Y_{q+1})| \leq |E_G(X_q, Y_q)| + |E_G(A_q, B_q)| \leq \frac{1}{100}|Y_q| + \frac{1}{100}|B_q| \leq \frac{1}{100}|Y_{q+1}|$. If $|X_{q+1}| \geq \frac{3n}{4}$, then we terminate the current phase and continue to Phase $(q+1)$. Otherwise, we return the cut $(A, B) = (X_{q+1}, Y_{q+1})$. In the latter case, we are guaranteed that $|A|, |B| \geq \frac{n}{4}$, and $|E_G(A, B)| \leq \frac{1}{100}|Y_{q+1}| \leq \frac{n}{100}$.

In the second case, the algorithm from Theorem 2.5 computes a routing $\mathcal{P}_i$ in G'_q of a subset $M'_i \subseteq M_i$ containing at least $(1-\alpha)|M_i| = |M_i| - \frac{\rho}{2}$ pairs of vertices, such that the total congestion caused by the paths in $\mathcal{P}_i$ is at most $\frac{2^{O(1/\epsilon^6)} \cdot \log^2 n}{\varphi^2} \leq 2^{O(1/\epsilon^6)} \cdot \Delta^2 \cdot \log^2 n$. In this case, we say that iteration i ended with a routing. For every edge $e \in M'_i$, we let $P(e) \in \mathcal{P}_i$ be the embedding path of edge e. We set $F_i = M_i \setminus M'_i$, so $|F_i| \leq \frac{\rho}{2}$, and we continue to the next iteration. This concludes the description of the ith iteration. We now complete the description of Phase q.

If any iteration in Phase q ended with a cut, then the phase is terminated as described above. We assume therefore from now on that every iteration in Phase q ended with a routing. Let $F = \bigcup_{i=1}^{19} F_i$. Recall that, for all $1 \leq i \leq 19$, $|F_i| \leq \rho$, so $|F| \leq 19\rho$. Consider the graph $G''_q = G'_q \cup F$, where we treat the edges of F as fake edges. For every fake edge $e \in F$, we let $P(e) = \{e\}$ be a path that embeds the edge e into itself in graph G''_q. Note that the maximum vertex degree in G''_q is at most $\Delta + 19 \leq 19\Delta$. We have also now obtained an embedding $\mathcal{P}'' = \{P(e) \mid e \in E(H_q)\}$ of H_q into G''_q, such that the paths in $\mathcal{P}''$ cause congestion η, where $\eta \leq 2^{O(1/\epsilon^6)} \cdot \Delta^2 \cdot \log^2 n$.

For convenience, we denote the maximum vertex degree of G''_q by $\Delta_G \leq 19\Delta$, the maximum vertex degree of H_q by $\Delta_H \leq 9$, and $\psi = \alpha_0$. Notice that:

$$\frac{\psi \cdot n_q}{32\Delta_G \eta} \geq \frac{\alpha_0 \cdot n}{2^{O(1/\epsilon^6)} \cdot \Delta^3 \cdot \log^2 n} \geq \frac{n}{2^{O(1/\epsilon^6)} \cdot \Delta^3 \cdot \log^2 n} \geq 20\rho \geq |F|,$$

since $\rho = \frac{n}{2^{\hat{c}/\epsilon^6} \cdot \Delta^3 \cdot \log^2 n}$, and $\hat{c}$ is a large enough constant. We can then use the algorithm AlgExtractExpander from Lemma 3.5 to compute a subgraph $G^* \subseteq G'_q$, such that G^* is a ψ'-expander, for $\psi' \geq \frac{\psi}{6\Delta_G \cdot \eta} \geq \frac{1}{2^{O(1/\epsilon^6)} \cdot \Delta^3 \cdot \log^2 n}$. Moreover, since $|F| \leq \frac{\psi \cdot n_q}{32\Delta_G \eta}$, we get that $\frac{4|F|\eta}{\psi} \leq \frac{n_q}{8\Delta_G}$, and so $|V(G^*)| \geq n_q - \frac{4|F|\eta}{\psi} \geq \frac{15n_q}{16}$. Since $n_q \geq \frac{5n}{8}$, we get that $|V(G^*)| \geq \frac{n}{2}$. We return the set $S = V(G^*)$ of vertices and terminate the algorithm.

We now bound the running time of a single phase. A phase has at most 19 iterations, and in every iteration we use the algorithm from Theorem 2.5, whose running time is bounded by $O\left(\frac{m^{1+O(\epsilon)}}{\varphi^3}\right) \leq O\left(m^{1+O(\epsilon)} \cdot \Delta^3\right)$. Additionally, the running time of the algorithm from Lemma 3.5 is bounded by $\tilde{O}(|E(G)|\Delta_G \cdot \eta/\psi) \leq \tilde{O}\left(n \cdot \Delta^4 \cdot 2^{O(1/\epsilon^6)}\right) \leq O\left(n^{1+O(\epsilon)} \cdot \Delta^4\right)$ (since $\epsilon > \frac{2}{(\log n)^{1/25}}$, so $n \geq 2^{(2/\epsilon)^{25}}$). Overall, the running time of a single phase is bounded by $O\left(m^{1+O(\epsilon)} \cdot \Delta^4\right)$.

Next, we bound the number of phases. Note that in every phase, the cardinality of the set Y_q of vertices grows by at least $\frac{\rho}{32} \geq \frac{n}{2^{\Theta(1/\epsilon^6)} \cdot \Delta^3 \cdot \log^2 n}$. Therefore, the number of phases is bounded by

$2^{O(1/\epsilon^6)} \cdot \Delta^3 \cdot \log^2 n$, and so the total running time of the algorithm is bounded by:

$$O\left(m^{1+O(\epsilon)} \cdot \Delta^7 \cdot 2^{O(1/\epsilon^6)}\right) \leq O\left(m^{1+O(\epsilon)} \cdot \Delta^7\right).$$

11 Further Applications

In this section we provide our improved deterministic approximation algorithms for Sparsest Cut, Lowest Conductance Cut, Minimum Balanced Cut, and Most-Balanced Sparse Cut. We also provide a new algorithm for expander decompositions. Several (but not all) of these results are obtained in the same manner as their weaker counterparts from [CGL+20], by plugging in our stronger algorithm for the Cut Player in the Cut-Matching Game from Theorem 2.6 instead of its weaker analogue from [CGL+20]. Some of the proofs in this section are therefore essentially identical to the proofs from [CGL+20], and are only provided here for completeness. We point out explicitly when this is the case. We start by introducing some technical tools that will be useful for us.

11.1 Main Technical Tools

In this subsection we introduce two main technical tools that will be used in order to obtain improved algorithms for Minimum Balanced Cut, Sparsest Cut, Lowest Conductance Cut, and Expander Decomposition. Both these tools - degree reduction and a faster algorithm for basic path peeling - appeared in [CGL+20], and we do not make any changes to them.

11.1.1 Degree Reduction

Some of our algorithms are easier to describe, and provide better guarantees, when the maximum vertex degree of the input graph is low. However, in general, an input graph may have an arbitrarily large maximum vertex degree. We describe here a standard algorithm for transforming a general graph into a low-degree graph, by replacing every vertex of the input graph with an expander of appropriate size. The algorithm is identical to that from [CGL+20], and similar algorithms have been used in the past extensively.

We now turn to describe a deterministic algorithm, that we call ReduceDegree. The algorithm is given as input an arbitrary graph $G = (V, E)$, and transforms it into a bounded-degree graph $\hat{G}$. Throughout, we denote $|V| = n$ and $|E| = m$. For convenience, we denote $V = \{v_1, \ldots, v_n\}$. For every vertex $v_i \in V$, we denote by $d(v_i)$ the degree of v_i in G, and we let $\left\{e_1(v_i), \ldots, e_{d(v_i)}(v_i)\right\}$ be the set of edges incident to v, indexed in an arbitrary order. For every vertex $v_i \in V$, we use Algorithm ConstructExpander from 3.3 to construct a graph H_i, whose vertex set $V_i = \left\{u_1(v_i), \ldots, u_{d(v_i)}(v_i)\right\}$ contains $d(v_i)$ vertices, such that H_i is an α_0-expander, and the maximum vertex degree in H_i is at most 9. Recall that the running time of the algorithm for constructing H_i is bounded by $O(d(v_i))$.

In order to obtain the final graph $\hat{G}$, we start with a disjoint union of all graphs in $\{H_i \mid v_i \in V\}$. All edges lying in such graphs H_i are called *type-1 edges*. Additionally, we add to $\hat{G}$ a collection of type-2 edges, defined as follows. Consider any edge $e = (v, v') \in E$, and assume that $e = e_j(v) = e_{j'}(v')$ (that is, e is the jth edge incident to v and it is the j'th edge incident to v'). We then let $\hat{e}$ be the edge $(u_j(v), u_{j'}(v))$. For every edge $e \in E$, we add the corresponding new edge $\hat{e}$ to graph $\hat{G}$ as a type-2 edge. This concludes the construction of the graph $\hat{G}$, that we denote by $\hat{G} = (\hat{V}, \hat{E})$. Note that the maximum vertex degree in $\hat{G}$ is at most 10, and $|\hat{V}| = 2m$. Moreover, the running time of the algorithm for constructing the graph $\hat{G}$ is $O(m)$.

We say that a set $S \subseteq \hat{V}$ of vertices of $\hat{G}$ is *canonical* if, for every vertex $v_i \in V$, either $V_i \subseteq S$, or $V_i \cap S = \emptyset$. Similarly, we say that a cut (X, Y) in a subgraph of $\hat{G}$ is canonical, if each of X, Y is a canonical subset of $\hat{V}$. The following lemma, that was proved in [CGL+20] allows us to convert an arbitrary sparse balanced cuts in a subgraph of $\hat{G}$ into a canonical one.

Lemma 11.1 (Lemma 5.4 from [CGL+20]) *Let $\alpha_0 > 0$ be the constant from Theorem 3.3. There is a deterministic algorithm, that we call* MakeCanonical*, that, given a subgraph $\hat{G}' \subseteq \hat{G}$, where $V(\hat{G}')$ is a canonical vertex set, together with a cut (A, B) in $\hat{G}'$, computes a canonical cut (A', B') in $\hat{G}'$, such that $|A'| \geq |A|/2$, $|B'| \geq |B|/2$, and moreover, if $|E_{\hat{G}}(A, B)| \leq \frac{\alpha_0}{2} \cdot \min\{|A|, |B|\}$, then $|E_{\hat{G}}(A', B')| \leq O(|E_{\hat{G}}(A, B)|)$. The running time of the algorithm is $O(m)$,*

11.1.2 Faster Basic Path Peeling

The following theorem provides a more efficient algorithm for basic Path Peeling. The theorem was proved in [CGL+20]; a similar result appeared in [NS17] (see Lemma B.18).

Theorem 11.2 (Theorem 7.1 from [CGL+20]) *There is a deterministic algorithm, that we call* MatchOrCut*, whose input consists of an m-edge graph $G = (V, E)$, two disjoint subsets A, B of its vertices with $|A| \leq |B|$, and parameters $z \geq 0$ and $0 < \varphi < 1/2$. The algorithm computes one of the following:*

- *either a matching $M \subseteq A \times B$ with $|M| > |A| - z$, such that there exists a set $\mathcal{P} = \{P(a, b) \mid (a, b) \in M\}$ of paths in G, where for each pair $(a, b) \in M$, path $P(a, b)$ connects a to b, and the paths in $\mathcal{P}$ cause congestion at most $O\left(\frac{\log n}{\varphi}\right)$; or*
- *a cut (X, Y) in G, with $|X|, |Y| \geq z/2$, and $|E_G(X, Y)| \leq \varphi \cdot \min\{|X|, |Y|\}$.*

The running time of the algorithm is $O\left(m^{1+o(1)}\right)$.

We note that the algorithm from Theorem 11.2 does not compute the set $\mathcal{P}$ of paths explicitly, as even listing all paths in the set may take time that is greater than $m^{1+o(1)}$, if parameter φ is sufficiently small. It only guarantees that set $\mathcal{P}$ of paths with the above properties exists.

11.2 Most-Balanced Sparse Cut

Recall that, given a cut (X, Y) in a graph G, the *sparsity* of the cut is $\frac{|E_G(X,Y)|}{\min\{|X|,|Y|\}}$. We sometimes refer to $\min\{|X|, |Y|\}$ as the *size* of the cut (X, Y).

In the Most Balanced Sparse Cut problem, the input is an n-vertex graph G, and a parameter $0 < \varphi \leq 1$. The goal is to compute a cut (X, Y) in G of sparsity at most φ, while maximizing the size $\min\{|X|, |Y|\}$ of the cut. An (α, β)-bicriteria approximation algorithm for the problem, given parameters $0 < \varphi < 1$ and $z \geq 1$, must either compute a cut (X, Y) in G of sparsity at most φ and size at least z; or correctly establish that every cut (X', Y') whose sparsity is at most φ/α has size at most $\beta \cdot z$.

In [CGL+20] (see Lemma 7.3), an (α, β)-bicriteria deterministic approximation algorithm was obtained for the Most Balanced Sparse Cut problem, with $\alpha = (\log n)^{O(1/\epsilon)}$ and $\beta = (\log n)^{O(1/\epsilon)}$, in time $O\left(m^{1+O(\epsilon)+o(1)} \cdot (\log n)^{O(1/\epsilon^2)}\right)$ for any $\frac{1}{c \log n} \leq \epsilon \leq 1$, for some fixed constant c. By using our algorithm for the Cut Player from Theorem 2.6, we immediately obtain the following stronger bicriteria approximation algorithm for the problem. The proof is essentially identical to the proof of Lemma

7.3 in [CGL+20]; the only difference is that we use the stronger algorithm for the Cut Player from Theorem 2.6. We provide the proof here for completeness.

Theorem 11.3 *There is a constant c_0, and a deterministic algorithm, that, given an n-vertex and m-edge graph $G=(V,E)$ and parameters $0<\varphi\leq 1$, $0<z\leq n$, and parameter $\frac{4}{(\log n)^{1/25}}<\epsilon<\frac{1}{400}$:*

- *either returns a cut (X,Y) in G with $|E_G(X,Y)|\leq \varphi\cdot\min\{|X|,|Y|\}$ and $|X|,|Y|\geq z$;*
- *or correctly establishes that, for every cut (X',Y') in G with $|E_G(X',Y')|\leq \frac{\varphi}{\alpha}\cdot\min\{|X'|,|Y'|\}$, $\min\{|X'|,|Y'|\}<\alpha'\cdot z$ holds, for $\alpha=2^{c_0/\epsilon^6}\cdot\log^7 n$ and $\alpha'=2^{c_0/\epsilon^6}\cdot\log^6 n$.*

The running time of the algorithm is $O\left(m^{1+O(\epsilon)+o(1)}\right)$.

Proof: The algorithm employs the Cut-Matching Game. We will maintain a set F of fake edges that are added to graph G. Initially, $F=\emptyset$. We assume that n is an even integer; otherwise we add a new isolated vertex v_0 to G, and we add a fake edge connecting v_0 to an arbitrary vertex of G to F. We also maintain a graph H, that initially contains the set V of vertices and no edges. We then perform a number of iterations, that correspond to the Cut-Matching Game. In every iteration i, we will add a matching M_i to graph H. We will ensure that the number of iterations is bounded by $O(\log n)$, so the maximum vertex degree in H is always bounded by $\Delta_H\leq O(\log n)$. At the beginning of the algorithm, graph H contains the set V of vertices and no edges. We now describe the execution of the ith iteration.

In order to execute the ith iteration, we apply Algorithm from Theorem 2.6 to the current graph H, with parameter ϵ remaining unchanged. Assume first that the output of the algorithm from Theorem 2.6 is a cut (A_i,B_i) in H with $|A_i|,|B_i|\geq n/4$ and $|E_H(A,B)|\leq n/100$. We treat this partition as the move of the Cut Player. Assume w.l.o.g. that $|A_i|\leq|B_i|$. Next, we compute an arbitrary partition (A_i',B_i') of $V(G)$ with $|A_i'|=|B_i'|$, such that $A_i\subseteq A_i'$. We apply Algorithm MatchOrCut from Theorem 11.2 to the sets A_i',B_i' of vertices, a sparsity parameter $\varphi'=\varphi/2$ and parameter $z'=4z$. If the algorithm returns a cut (X,Y) in G, with $|X|,|Y|\geq z'/2\geq 2z$, and $|E_G(X,Y)|\leq\varphi'\cdot\min\{|X|,|Y|\}$, then we terminate the algorithm and return the cut (X,Y), after we delete the extra vertex v_0 from it (if it exists). It is easy to verify that $|X|,|Y|\geq z$ and that $|E_G(X,Y)|\leq\varphi\cdot\min\{|X|,|Y|\}$. Otherwise, the algorithm from Theorem 11.2 computes a matching $M_i'\subseteq A_i'\times B_i'$ with $|M_i'|\geq|A_i'|-4z$, such that there exists a set $\mathcal{P}_i'=\{P(a,b)\mid(a,b)\in M_i'\}$ of paths in G, where for each pair $(a,b)\in M_i'$, path $P(a,b)$ connects a to b, and the paths in $\mathcal{P}_i'$ cause congestion at most $O\left(\frac{\log n}{\varphi}\right)$. We let $A_i''\subseteq A_i'$, $B_i''\subseteq B_i'$ be the sets of vertices that do not participate in the matching M_i', and we let M_i'' be an arbitrary perfect matching between these vertices. We define a set F_i of fake edges, containing the edges of M_i'', and an embedding $\mathcal{P}_i''=\{P(e)\mid e\in F_i\}$ of the edges in M_i'', where each fake edge is embedded into itself. Lastly, we set $M_i=M_i'\cup M_i''$. We view the matching M_i as the response of the matching player in the Cut-Matching Game. We add the edges of M_i to H, and continue to the next iteration. Notice that $|F_i|\leq 4z$.

We perform the iterations as described above, until the algorithm from Theorem 2.6 returns a subset $S\subseteq V$ of at least $n/2$ vertices, such that graph $H[S]$ is φ^*-expander, for $\varphi^*\geq\Omega\left(\frac{1}{2^{O(1/\epsilon^6)}\cdot\Delta_H^3\cdot\log^2 n}\right)\geq\Omega\left(\frac{1}{2^{O(1/\epsilon^6)}\cdot\log^5 n}\right)$. Recall that Theorem 3.6 guarantees that this must happen after at most $O(\log n)$ iterations. We then perform one last iteration, whose index we denote by q.

We let $B_q=S$ and $A_q=V(G)\setminus S$, and apply Algorithm MatchOrCut from Theorem 11.2 to the sets A_q,B_q of vertices, a sparsity parameter $\varphi'=\varphi/2$ and parameter $z'=4z$. As before, if the

algorithm returns a cut (X,Y) in G, with $|X|,|Y| \geq z'/2 \geq 2z$ and $|E_G(X,Y)| \leq \varphi' \cdot \min\{|X|,|Y|\}$, then we terminate the algorithm and return the cut (X,Y), after we delete the extra vertex v_0 from it (if it exists). As before, we get that $|X|,|Y| \geq z$ and $|E_G(X,Y)| \leq \varphi \cdot \min\{|X|,|Y|\}$. Otherwise, the algorithm from Theorem 11.2 computes a matching $M'_q \subseteq A'_q \times B'_q$ with $|M'_q| \geq |A_q| - 4z$, such that there exists a set $\mathcal{P}'_q = \left\{P(a,b) \mid (a,b) \in M'_q\right\}$ of paths in G, where for each pair $(a,b) \in M'_q$, path $P(a,b)$ connects a to b, and the paths in $\mathcal{P}'_q$ cause congestion at most $O\left(\frac{\log n}{\varphi}\right)$. We let $A'_q \subseteq A_q$, $B'_q \subseteq B_q$ be the sets of vertices that do not participate in the matching M'_q, and we let M''_q be an arbitrary matching that connects every vertex of A'_q to a distinct vertex of B'_q (such a matching must exist since $|A_q| \leq |B_q|$). As before, we define a set F_q of fake edges, containing the edges of M''_q, and an embedding $\mathcal{P}''_q = \{P(e) \mid e \in F_q\}$ of the edges in M''_q, where each fake edge is embedded into itself. Lastly, we set $M_q = M'_q \cup M''_q$, and we add the edges of M_q to graph H.

From now on we assume that the algorithm never terminated with a cut (X,Y) with $|X|,|Y| \geq z$ and $|E_G(X,Y)| \leq \varphi \cdot \min\{|X|,|Y|\}$. Note that, from Observation 3.2, the final graph H is a ψ-expander, for $\psi \geq \frac{\varphi^*}{2} \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$. Moreover, we are guaranteed that there is an embedding of H into $G+F$ with congestion $O\left(\frac{\log^2 n}{\varphi}\right)$, where $F = \bigcup_{i=1}^{r} F_i$ is a set of $O(z \log n)$ fake edges. Notice that, in the embedding that we constructed, every edge of H is either embedded into a path consisting of a single fake edge, or it is embedded into a path in the graph G; every fake edge in F serves as an embedding of exactly one edge of H.

We now claim that there is a large enough universal constant c_0, such that, if we let $\alpha = 2^{c_0/\epsilon^6} \cdot \log^5 n$ and $\alpha' = 2^{c_0/\epsilon^6} \cdot \log^6 n$, then for every cut (X',Y') in G with $|E_G(X',Y') \leq \frac{\varphi}{\alpha} \cdot \min\{|X'|,|Y'|\}$, $\min\{|X'|,|Y'|\} < \alpha' \cdot z$ holds.

Indeed, consider any cut (X',Y') in G with $|X'|,|Y'| \geq \alpha' \cdot z$. We assume w.l.o.g. that $|X'| \leq |Y'|$. It is enough to show that $|E_G(X',Y')| > \frac{\varphi \cdot |X'|}{\alpha}$.

Notice that (X',Y') also defines a cut in graph H, and, since H is a ψ-expander, $|E_H(X',Y')| \geq \psi \cdot |X'| \geq \psi \cdot \alpha' \cdot z \geq \psi \cdot z \cdot 2^{c_0/\epsilon^6} \cdot \log^6 n$. Since $\psi \geq \frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}$, assuming that c_0 is a large enough constant, we get that $|E_H(X',Y')| \geq c_0 z \log n$.

We partition the set $E_H(X',Y')$ of edges into two subsets. The first subset, E_1, is the set of edges corresponding to the fake edges (so each edge $e \in E_1$ is embedded into a path $P(e) = \{e\}$ in $G+F$), and E_2 contains all remaining edges (each of which is embedded into a path of G). Recall that the total number of the fake edges, $|F| \leq O(z \log n)$, while $|E_H(X',Y')| \geq c_0 z \log n$. Therefore, by letting c_0 be a large enough constant, we can ensure that $|E_1| \leq |E_H(X',Y')|/2$.

The embedding of H into $G+F$ defines, for every edge $e \in E_2$ a corresponding path $P(e)$ in G, that must contribute at least one edge to the cut $E_G(X',Y')$. Since the embedding causes congestion $O\left(\frac{\log^2 n}{\varphi}\right)$, we get that:

$$\begin{aligned}|E_G(X',Y')| &\geq \Omega\left(\frac{|E_H(X',Y')|\cdot\varphi}{\log^2 n}\right)\\ &\geq \Omega\left(\frac{\varphi\cdot\psi\cdot|X'|}{\log^2 n}\right)\\ &\geq \Omega\left(\frac{\varphi\cdot|X'|}{2^{O(1/\epsilon^6)}\cdot\log^7 n}\right)\\ &> \frac{\varphi\cdot|X'|}{2^{c_0/\epsilon^6}\cdot\log^7 n}\\ &= \frac{\varphi|X'|}{\alpha},\end{aligned}$$

(we have used the fact that c_0 is a large enough constant). We note that we have ignored the extra vertex v_0 that we have added to G if $|V(G)|$ is odd, but the removal of this vertex can only change the cut sparsity and the cardinalities of X' and Y' by a small constant factor that can be absorbed in c_0.

Lastly, we bound the running time of the algorithm. The algorithm consists of $O(\log n)$ iterations. Every iteration employs the algorithm from Theorem 2.6, whose running time is $O\left(|E(H)|^{1+O(\epsilon)}\cdot\Delta_H^7\right)\leq O\left(n^{1+O(\epsilon)}\right)$, since $\Delta_H\leq O(\log n)$, and $\log^8 n < n^{4\epsilon}$ (the latter follows from the assumption that $\epsilon > \frac{2}{(\log n)^{1/25}}$, and since, from the inequality $\frac{2}{(\log n)^{1/25}} < \epsilon < \frac{1}{400}$, n must be large enough). Additionally, in every iteration we use Algorithm MatchOrCut from Theorem 11.2, whose running time is $O\left(m^{1+o(1)}\right)$. Therefore, the total running time is $O\left(m^{1+O(\epsilon)+o(1)}\right)$. □

We also immediately obtain the following analogue of Lemma 7.4 from [CGL+20].

Theorem 11.4 *There is a constant c_0 and a deterministic algorithm, that, given an n-vertex and m-edge graph $G=(V,E)$ and parameters $0<\varphi\leq 1$ and $\frac{4}{(\log n)^{1/25}} < \epsilon < \frac{1}{400}$:*

- *either returns a cut (X,Y) in G with $|E_G(X,Y)|\leq\varphi\cdot\min\{|X|,|Y|\}$;*
- *or correctly establishes that G is a φ'-expander, for $\varphi'=\frac{\varphi}{2^{c_0/\epsilon^6}\cdot\log^7 n}$.*

The running time of the algorithm is $O\left(m^{1+O(\epsilon)+o(1)}\right)$.

Proof: The proof is almost identical to the proof of Theorem 11.3. The only difference is that we set the parameter z that is used in the calls to Algorithm MatchOrCut from 11.2 to 1. This ensures that no fake edges are introduced. □

11.3 Sparsest Cut and Lowest-Conductance Cut – Proof of Theorem 2.7

In this section we prove Theorem 2.7. Theorem 11.4 immediately gives a deterministic $\left(2^{O(1/\epsilon^6)}\cdot\log^7 n\right)$-approximation algorithm for the Sparsest Cut problem with running time $O\left(m^{1+O(\epsilon)+o(1)}\right)$, for all $\epsilon\geq\frac{4}{(\log n)^{1/25}}$. Indeed, let G be the input m-edge graph. For $1\leq i\leq\lceil\log m\rceil$, let $\varphi_i=1/2^i$. For $1\leq i\leq\lceil\log m\rceil$, we apply the algorithm from Theorem 11.4 to graph G, with the parameter φ_i. Let i be the smallest integer, for which the algorithm returned a cut (X,Y) with $|E_G(X,Y)|<\varphi_i\cdot\min|X|,|Y|$. Then, when applied to G with parameter $\varphi_{i+1}=\varphi_i/2$, the algorithm correctly established that G is a φ'-expander, for $\varphi'=\frac{\varphi_i}{2^{O(1/\epsilon^6)}\cdot\log^7 n}$. In other words, the sparsity of the sparsest cut in G is at least φ'.

Therefore, (X,Y) is a $2^{O(1/\epsilon^6)} \cdot \log^7 n$-approximate sparsest cut. The running time of the algorithm remains $O\left(m^{1+O(\epsilon)+o(1)}\right)$. By setting $\epsilon = (1/c \log\log\log n)^{1/6}$, for a large enough constant c, we obtain a factor-$O(\log^7 n \log\log n)$-approximation, in time $O\left(m^{1+o(1)}\right)$.

We now show that we can obtain an algorithm with similar guarantees for the Lowest Conductance Cut problem. The algorithm follows easily from the algorithm for the Sparsest Cut problem, and is identical to that of [CGL+20]. The only difference is that we are using the stronger algorithm for Sparsest Cut that we obtained. Let $G=(V,E)$ be an input to the Lowest Conductance Cut problem, with $|V|=n$ and $|E|=m$. Let $\frac{4}{(\log n)^{1/25}} < \epsilon < \frac{1}{400}$ be a parameter. We start by obtaining a factor-$\left(2^{O(1/\epsilon^6)} \cdot \log^7 n\right)$-approximation algorithm with running time $O\left(m^{1+O(\epsilon)+o(1)}\right)$.

Denote by ψ the conductance of the lowest-conductance cut in G. We can assume without loss of generality that $\psi < \frac{1}{2^{c/\epsilon^6} \cdot \log^7 n}$ for some large enough constant c, since otherwise we can let v be a lowest-degree vertex in G, and return the cut $(\{v\}, V\setminus\{v\})$, whose conductance is 1. We use Algorithm ReduceDegree from Section 11.1.1, in order to construct, in time $O(m)$, a graph $\hat{G}$, whose maximum vertex degree is bounded by 10, and $|V(\hat{G})| = 2m$.

Note that, if we denote φ the value of the sparsest cut in $\hat{G}$, then $\varphi \le \psi$ must hold. This is since every cut (A,B) in G naturally defines a cut (A',B') in $\hat{G}$, with $|A'| = \mathrm{Vol}_G(A), |B'| = \mathrm{Vol}_G(B)$, and $|E_{\hat{G}}(A',B')| = |E_G(A,B)|$. We use our approximation algorithm for the Sparsest Cut problem in graph $\hat{G}$, to obtain a cut (X',Y') of $\hat{G}$, whose sparsity is at most $\left(2^{O(1/\epsilon^6)} \cdot \log^7 n\right) \cdot \varphi$, in time $O\left(m^{1+O(\epsilon)+o(1)}\right)$.

Using Algorithm MakeCanonical from Lemma 11.1, we obtain a cut (X'',Y'') of $\hat{G}$, with $|X''| \ge |X'|/2$, $|Y''| \ge |Y'|/2$, and $|E_{\hat{G}}(X'',Y'')| \le O(|E_{\hat{G}}(X',Y')|) \le \left(2^{O(1/\epsilon^6)} \cdot \log^7 n\right) \cdot \varphi \cdot \min\{|X'|,|Y'|\} \le \left(2^{O(1/\epsilon^6)} \cdot \log^7 n\right) \cdot \psi \cdot \min\{|X''|,|Y''|\}$, such that both X'' and Y'' are canonical vertex sets. This cut naturally defines a cut (X,Y) in G, with $\mathrm{Vol}_G(X) = |X''|$, $\mathrm{Vol}_G(Y) = |Y''|$, and $|E_G(X,Y)| = |E_{\hat{G}}(X'',Y'')|$. Therefore:

$$\begin{aligned}
|E_G(X,Y)| &= |E_{\hat{G}}(X'',Y'')| \\
&\le \left(2^{O(1/\epsilon^6)} \cdot \log^7 n\right) \cdot \psi \cdot \min\left\{|X''|,|Y''|\right\} \\
&\le \left(2^{O(1/\epsilon^6)} \cdot \log^7 n\right) \cdot \psi \cdot \min\left\{\mathrm{Vol}_G(X), \mathrm{Vol}_G(Y)\right\}.
\end{aligned}$$

We conclude that cut (X,Y) is a factor $(2^{O(1/\epsilon^6)} \cdot \log^7 n)$-approximate solution to instance G of Lowest Conductance Cut. Since the running time of Algorithm MakeCanonical is $O(m)$, the running time of the whole algorithm remains bounded by $O\left(m^{1+O(\epsilon)+o(1)}\right)$. As before, by setting $\epsilon = (1/c \log\log\log n)^{1/6}$, for a large enough constant c, we obtain a factor-$O(\log^7 n \log\log n)$-approximation for Lowest Conductance Cut, in time $O\left(m^{1+o(1)}\right)$.

11.4 Minimum Balanced Cut – Proof of Theorems 2.8 and 2.9

In this section we prove Theorem 2.8 and Theorem 2.9. Our proof is somewhat more involved than that of [CGL+20], who iteratively used the algorithm for Most Balanced Sparse Cut from (the weaker version of) Theorem 11.3. The reason is that, while our bounds for the parameters α and α' are better than those obtained by Theorem 11.3, they are still super-logarithmic, and if we follow the framework

of [CGL+20], who apply the algorithm from Theorem 11.3 over the course of $O(1/\epsilon)$ iterations, we will still accumulate an approximation factor that is at least as high as $(\log n)^{\Theta(1/\epsilon)}$.

The proofs of Theorem 2.8 and Theorem 2.9 are very similar to each other, and we start with presenting the part that is common to both proofs. Recall that we are given as input a graph G with $|V(G)| = n$, $|E(G)| = m$ and a parameter $0 < \psi \leq 1$. We can assume w.l.o.g. that both m and n are greater than a large enough constant, since otherwise we can use the algorithm of [CGL+20], whose running time can now be bounded by $O(m)$.

We use a parameter $\epsilon = 1/(\log\log\log m)^{1/25}$. It is easy to verify that $m^\epsilon = 2^{O(\log n/(\log\log\log n)^{1/25})} > \log^4 n$ must hold. We can also assume that $\psi < 1/(2^{c/\epsilon^6}\log^8 n)$ for a large enough constant c, since otherwise we can compute an arbitrary partition (A, B) of $V(G)$ with $\mathrm{Vol}_G(A), \mathrm{Vol}_G(A) \geq \mathrm{Vol}(G)/3$ (since for every vertex $v \in V(G)$, $\deg_G(v) < \mathrm{Vol}(G)/2$, such a partition can be computed by a simple greedy algorithm, that iteratively adds each vertex to a set of $\{A, B\}$, whose current volume is smaller). Clearly, $|E_G(A, B)| \leq m \leq \frac{\mathrm{Vol}(G)}{2} \leq \psi \cdot 2^{c/\epsilon^6} \cdot (\log^8 n) \cdot \mathrm{Vol}(G) \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$. Therefore, from now on we assume that $\psi < 1/(2^{c/\epsilon^6}\log^8 n)$ for a large enough constant c.

As in the algorithm of [CGL+20], we start by applying Algorithm ReduceDegree from Section 11.1.1 to graph G, in order to construct, in time $O(m)$, a graph $\hat{G}$ whose maximum vertex degree is bounded by 10, and $|V(\hat{G})| = 2m$. Denote $V(G) = \{v_1, \ldots, v_n\}$. Recall that graph $\hat{G}$ is constructed from graph G by replacing each vertex v_i with an α_0-expander $H(v_i)$ on $\deg_G(v_i)$ vertices, where $\alpha_0 = \Theta(1)$. For convenience, we denote the set of vertices of $H(v_i)$ by V_i. Therefore, $V(\hat{G}) = V_1 \cup V2 \cup \cdots \cup V_n$. Denote $|V(\hat{G})| = \hat{n}$. Consider now some subset S of vertices of $\hat{G}$. As before, we say that S is a *canonical* set of vertices if, for all $1 \leq i \leq n$, either $V_i \subseteq S$ or $V_i \cap S = \emptyset$ holds. Our starting point is the following lemma, that is an easy application of the Cut-Matching Game, combined with Algorithm MakeCanonical from Lemma 11.1. The proof is included in Section B of Appendix.

Lemma 11.5 *There is a deterministic algorithm, whose input consists of a canonical set $V' \subseteq V(\hat{G})$ of vertices of $\hat{G}$, with $|V'| \geq 2\hat{n}/3$, and parameters $0 < \varphi < 1$ and $\rho \geq \log n$. The algorithm computes one of the following:*

- *either a partition (X, Y) of V', where both X, Y are canonical subsets of $V(\hat{G})$, $|X|, |Y| \geq \rho$, and $|E_{\hat{G}}(X, Y)| \leq \varphi \cdot \min\{|X|, |Y|\}$;*
- *or a φ^*-expander graph H with $V(H) = V'$ and maximum vertex degree $O(\log n)$, where $\varphi^* \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$, together with a set F of at most $O(\rho \cdot \log n)$ edges of H, such that there exists an embedding $\mathcal{P}$ of $H \setminus F$ into $\hat{G}[V']$ with congestion at most $O\left(\frac{\log^2 n}{\varphi}\right)$.*

The running time of the algorithm is $O\left(m^{1+O(\epsilon)+o(1)}\right)$.

We emphasize that the algorithm from the above lemma does not compute the embedding $\mathcal{P}$ explicitly, and instead it only guarantees its existence. We obtain the following immediate corollary of Lemma 11.5. The corollary uses the parameter $\varphi^* \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$ from Lemma 11.5.

Corollary 11.6 *There is a deterministic algorithm, that, given a parameter $\rho \geq \log n$, and a parameter $c' > 1$, computes a cut (X^*, Y^*) in graph $\hat{G}$, such that both X^* and Y^* are canonical sets of vertices, $|X^*| \geq |Y^*|$, and $|E_{\hat{G}}(X^*, Y^*)| \leq \frac{c'\psi \log^3 n}{\varphi^*} \cdot \hat{n}$. Moreover, if $|Y^*| < \hat{n}/3$, then the algorithm also computes a φ^*-expander graph H with $V(H) = X^*$ and maximum vertex degree $O(\log n)$, together with a set F of at most $O(\rho \log n)$ edges of H, such that there exists an embedding $\mathcal{P}$ of $H \setminus F$ into $\hat{G}[X^*]$ with congestion $\eta \leq O\left(\frac{\varphi^*}{c'\psi \log n}\right)$. The running time of the algorithm is $O\left(\frac{m^{2+O(\epsilon)+o(1)}}{\rho}\right)$.*

Proof: The proof easily follows by iteratively applying the algorithm from Lemma 11.5. Let $\varphi = \frac{c'\psi \log^3 n}{\varphi^*}$. Our algorithm consists of at most $r = \frac{\hat{n}}{\rho}$ iterations. For all $1 \leq i \leq r$, at the beginning of iteration i, we are given a partition (X_i, Y_i) of $V(\hat{G})$, such that both X_i, Y_i are canonical sets, $|X_i| \geq \frac{2\hat{n}}{3}$, and $|E_{\hat{G}}(X_i, Y_i)| \leq \varphi \cdot |Y_i|$. At the beginning of the algorithm, we use the partition (X_1, Y_1) of $V(\hat{G})$, with $X_1 = V(\hat{G})$ and $Y_1 = \emptyset$. We now describe the execution of the ith iteration.

In order to execute the ith iteration, we apply the algorithm from Lemma 11.5 to set $V' = X_i$ of vertices of $\hat{G}$, and parameters ρ and φ, that remain unchanged. We now consider two cases.

Assume first, that the algorithm returned a partition (X', Y') of X_i, with $|X'|, |Y'| \geq \rho$, and $|E_{\hat{G}}(X', Y')| \leq \varphi \cdot \min\{|X'|, |Y'|\}$, such that both X' and Y' are canonical sets of vertices. Assume w.l.o.g. that $|X'| \geq |Y'|$. We then construct a new cut (X_{i+1}, Y_{i+1}) in $\hat{G}$, by letting $X_{i+1} = X'$ and $Y_{i+1} = V(\hat{G}) \setminus X' = Y' \cup Y_i$. Note that cut (X_{i+1}, Y_{i+1}) can be obtained from cut (X_i, Y_i) by moving the vertices of Y' from X_i to Y_i. Therefore, $E_{\hat{G}}(X_{i+1}, Y_{i+1}) \subseteq E_{\hat{G}}(X_i, Y_i) \cup E_{\hat{G}}(X', Y')$, and:

$$|E_{\hat{G}}(X_{i+1}, Y_{i+1})| \leq |E_{\hat{G}}(X_i, Y_i)| + |E_{\hat{G}}(X', Y')| \leq \varphi \cdot |Y_i| + \varphi \cdot |Y'| \leq \varphi \cdot |Y_{i+1}|.$$

If $|X_{i+1}| \geq 2\hat{n}/3$ holds, then we simply continue to the next iteration. Otherwise, since $|X_i| \geq 2\hat{n}/3$, we get that $|X_{i+1}| \geq |X_i|/2 \geq \hat{n}/3$. Therefore, we obtain a partition (X_{i+1}, Y_{i+1}) of $V(\hat{G})$ with $|X_{i+1}|, |Y_{i+1}| \geq \hat{n}/3$, and $|E_{\hat{G}}(X_{i+1}, Y_{i+1})| \leq \varphi \cdot |Y_{i+1}| \leq \varphi \cdot \hat{n} = \frac{c'\psi \log^3 n}{\varphi^*} \cdot \hat{n}$. We then return cut $(X^*, Y^*) = (X_{i+1}, Y_{i+1})$ and terminate the algorithm.

Next, we assume that the algorithm from Lemma 11.5 returned a φ^*-expander graph H with $V(H) = X_i$ and maximum vertex degree $O(\log n)$, together with a set F of at most $O(\rho \cdot \log n)$ edges of H, such that there exists an embedding $\mathcal{P}$ of $H \setminus F$ into $\hat{G}[X_i]$ with congestion at most $O\left(\frac{\log^2 n}{\varphi}\right) \leq O\left(\frac{\varphi^*}{c'\psi \log n}\right)$. In this case, we return cut $(X^*, Y^*) = (X_i, Y_i)$, graph H, and set F of edges.

It now remains to bound the running time of the algorithm. Notice that in every iteration, the cardinality of the set Y_i of vertices grows by at least ρ, and, since $|V(\hat{G})| = 2m$, the number of iterations is bounded by $2m/\rho$. In each iteration we use the algorithm from Lemma 11.5, whose running time is at most $O\left(m^{1+O(\epsilon)+o(1)}\right)$. Overall, the running time of the algorithm is bounded by $O\left(\frac{m^{2+O(\epsilon)+o(1)}}{\rho}\right)$. □

We are now ready to complete the proofs of Theorem 2.8 and Theorem 2.9.

11.4.1 Proof of Theorem 2.8

We start by computing a graph $\hat{G}$ exactly as described above, and then applying the algorithm from Corollary 11.6 to it, with parameter $\rho = \frac{\psi m}{\tilde{c}^2 \log n}$, where $\tilde{c}$ is a large constant, whose value we set later, and parameter c', that is a large enough constant, whose value we set later. Since $\psi > \frac{\log^3 m}{m}$, and m is large enough, we get that $\rho > \log n$. Let (X^*, Y^*) be the outcome of the algorithm.

Recall that both X^* and Y^* are canonical sets of vertices, $|X^*| \geq |Y^*|$, and $|E_{\hat{G}}(X^*, Y^*)| \leq \frac{c'\psi \log^3 n}{\varphi^*} \cdot \hat{n}$. Notice that cut (X^*, Y^*) in $\hat{G}$ naturally defines a cut (A, B) in G: for every vertex $v_i \in V(G)$, we add v_i to A if $V_i \subseteq X^*$, and we add it to B otherwise. It is immediate to verify that $\mathrm{Vol}_G(A) = |X^*|$, $\mathrm{Vol}_G(B) = |Y^*|$, and $|E_G(A, B)| = |E_{\hat{G}}(X^*, Y^*)| \leq \frac{c'\psi \log^3 n}{\varphi^*} \cdot \hat{n} \leq \psi \cdot 2^{O(1/\epsilon^6)} \cdot (\log^8 n) \cdot \mathrm{Vol}(G) \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$, since $\varphi^* \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$ and $\epsilon = 1/(\log\log\log m)^{1/25}$.

Consider first the case that $|Y^*| \geq \hat{n}/3$. Then, from the above discussion, $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \hat{n}/3 = \mathrm{Vol}(G)/3$. In this case, we return cut (A, B) as the outcome of the algorithm.

We assume from now on that $|Y^*| < \hat{n}/3$, and so the algorithm from Corollary 11.6 computed a φ^*-expander graph H with $V(H) = X^*$ and maximum vertex degree $O(\log n)$, together with a set F of at most $O(\rho \log n)$ edges of H, such that there exists an embedding $\mathcal{P}$ of $H \setminus F$ into $\hat{G}[X^*]$ with congestion at most η, where $\eta \leq O\left(\frac{\varphi^*}{c'\psi \log n}\right)$. Since $\rho = \frac{\psi m}{\tilde{c}^2 \log n}$, and $\tilde{c}$ is a sufficiently large constant, we get that $|F| \leq \frac{\psi m}{\tilde{c}}$. Let $\hat{G}'$ be the graph that is obtained from $\hat{G}[X^*]$, by adding the edges of F to it. We need the following simple observation.

Observation 11.7 *Graph $\hat{G}'$ is a φ'-expander, for $\varphi' = \Omega(c'\psi \log n)$.*

Proof: Consider any partition (A', B') of $V(\hat{G}')$, with $|A'| \leq |B'|$. It is enough to prove that $|E_{\hat{G}'}(A', B')| \geq \varphi' \cdot |A'|$. Recall that the set $\mathcal{P}$ of paths defines an embedding of $H \setminus F$ into $\hat{G}[X^*]$ with congestion at most η. By embedding every edge of F into itself, we can augment the set $\mathcal{P}$ of paths to obtain an embedding of H into $\hat{G}' = \hat{G}[X^*] \cup F$, with congestion at most η.

Notice that cut (A', B') in $\hat{G}'$ also defines a cut in graph H. Since graph H is a φ^*-expander, if we denote by $E' = E_H(A', B')$, then $|E'| \geq \varphi^* \cdot |A'|$. Consider now the set $\mathcal{P}' = \{P(e) \mid e \in E'\}$ of paths, where for each edge $e \in E'$, $P(e)$ is the path of $\mathcal{P}$ that serves as an embedding of the edge e. Then every path in $\mathcal{P}'$ must contain an edge of $E_{\hat{G}'}(A', B')$, and the paths in $\mathcal{P}'$ cause congestion at most η. Therefore, $|E_{\hat{G}'}(A', B')| \geq \frac{|E'|}{\eta} \geq \frac{\varphi^* \cdot |A'|}{\eta} \geq \Omega(c'\psi |A'| \log n) = \varphi' \cdot |A'|$, since $\eta \leq O\left(\frac{\varphi^*}{c'\psi \log n}\right)$. □

Recall that we have used the cut (X^*, Y^*) in $\hat{G}$ to define a cut (A, B) in G, with $\mathrm{Vol}_G(A) = |X^*|$, $\mathrm{Vol}_G(B) = |Y^*|$, and $|E_G(A, B)| = |E_{\hat{G}}(X^*, Y^*)| \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$. Consider now a graph G', that, intuitively, is obtained from $G[A]$, by adding the edges of F to it. Formally, in order to obtain graph G', we start from graph $G[A]$, and the consider the edges $e \in F$ one by one. Let $e = (x, y)$ be any such edge, and let $v_i, v_j \in A$ be the vertices with $x \in V_i$ and $y \in V_j$. If $i \neq j$, then we add edge $e' = (v_i, v_j)$ to graph G', and we think of e' as a *copy of edge* e. Notice that graph G' can be equivalently obtained from graph $\hat{G}'$ by contracting, for every vertex $v_i \in A$, all vertices of V_i into a single node. Next, we use the following easy observation:

Observation 11.8 *Graph G' has conductance at least 24ψ.*

Proof: Consider any cut (X, Y) in G'. We can naturally define a corresponding cut $(\hat{X}, \hat{Y})$ in graph $\hat{G}'$: for every vertex $v_i \in X$, we add all vertices of V_i to $\hat{X}$, and for every vertex $v_i \in Y$, we add all vertices of V_i to $\hat{Y}$. It is easy to verify that $|\hat{X}| = \mathrm{Vol}_G(X)$. Since every vertex of X^* is incident to at most $c^* \log n$ edges of F, for some constant c^* (as maximum vertex degree in H is $O(\log n)$), it is easy to verify that, for every vertex $v_i \in A$, $\deg_{G'}(v_i) \leq (c^* \log n) \deg_G(v_i)$. Therefore, $\mathrm{Vol}_{G'}(X) \leq (c^* \log n) \mathrm{Vol}_G(X)$, and so $|\hat{X}| \geq \frac{\mathrm{Vol}_{G'}(X)}{c^* \log n}$. Using similar reasoning, $|\hat{Y}| \geq \frac{\mathrm{Vol}_{G'}(Y)}{c^* \log n}$. Lastly, since, from Observation 11.7, graph $\hat{G}'$ is a φ'-expander, for $\varphi' = \Omega(c'\psi \log n)$, we get that $|E_{G'}(X, Y)| = |E_{\hat{G}'}(\hat{X}, \hat{Y})| \geq \varphi' \cdot \min\left\{|\hat{X}|, |\hat{Y}|\right\} \geq \Omega\left(\frac{c'\psi}{c^*}\right) \cdot \min\{\mathrm{Vol}_{G'}(X), \mathrm{Vol}_{G'}(Y)\}$. Since c^* is a fixed constant, and we can set c' to be a large enough constant, we get that $|E_{G'}(X, Y)| \geq 24\psi \cdot \min\{\mathrm{Vol}_{G'}(X), \mathrm{Vol}_{G'}(Y)\}$. We conclude that the conductance of graph G' is at least 24ψ. □

Next, we use the following pruning theorem of [SW19], that works with graph conductance instead of expansion.

Theorem 11.9 (Restatement of Theorem 1.3 from [SW19]) *There is a deterministic algorithm, that, given access to adjacency list of a graph $G = (V, E)$ that has conductance ψ, for some $0 < \psi \leq 1$, and a collection $E' \subseteq E$ of $k \leq \psi|E|/10$ edges of G, computes a subgraph $G' \subseteq G \setminus E'$, that has conductance at least $\psi/6$. Moreover, if we denote $A = V(G')$ and $B = V(G) \setminus A$, then $|E_G(A, B)| \leq 4k$, and $\mathrm{Vol}_G(B) \leq 8k/\psi$. The total running time of the algorithm is $\tilde{O}(k/\psi^2)$.*

We apply the algorithm from Theorem 11.9 to graph G', conductance parameter 24ψ, and the set $E' = F$ of edges. Since we have assumed that $|Y^*| < \hat{n}/3 = \mathrm{Vol}(G)/3$, we get that $|X^*| \geq 2\hat{n}/3 \geq 2\,\mathrm{Vol}(G)/3$. At the same time, $|E_G(X^*, Y^*)| < \psi \cdot 2^{O(1/\epsilon^6)} \cdot (\log^8 n) \cdot \mathrm{Vol}(G) < \mathrm{Vol}(G)/10$, since we have assumed that $\psi < \frac{1}{2^{c/\epsilon^6} \cdot (\log^8 n)}$ for a large enough constant c. Therefore, $|E(G')| \geq \frac{\mathrm{Vol}_G(X^*) - |E_G(X^*, Y^*)|}{2} \geq \frac{\mathrm{Vol}(G)}{4} \geq \frac{m}{2}$. Recall that $|F| \leq \frac{\psi m}{\tilde{c}}$ holds, where $\tilde{c}$ is a large enough constant. Therefore, $|F| \leq \frac{24\psi |E(G')|}{10}$.

The algorithm must then return a partition (Z, Z') of $V(G') = A$, such that graph $G[Z]$ has conductance at least $\frac{24\psi}{6} \geq \psi$, and $\mathrm{Vol}_{G'}(Z') \leq \frac{8|F|}{24\psi} \leq \frac{m}{3\tilde{c}}$, while $|E_G(Z, Z')| \leq 4|F| \leq \frac{4\psi m}{\tilde{c}} \leq O(\psi) \cdot \mathrm{Vol}(G)$.

We construct a cut (A^*, B^*) in graph G, by letting $A^* = Z$ and $B^* = Z' \cup Y^*$. Notice that $|E_G(A^*, B^*)| \subseteq |E_G(A, B)| + |E_G(Z, Z')| \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$. Additionally, we get that:

$$\mathrm{Vol}_G(A^*) = \mathrm{Vol}_G(Z) \geq \mathrm{Vol}_G(A) - \mathrm{Vol}_G(Z') \geq \frac{2\,\mathrm{Vol}(G)}{3} - \frac{m}{3\tilde{c}} \geq \frac{7\,\mathrm{Vol}(G)}{12}.$$

If $\mathrm{Vol}_G(B^*) \geq \mathrm{Vol}(G)/3$, then we get that $\mathrm{Vol}_G(A^*), \mathrm{Vol}_G(B^*) \geq \mathrm{Vol}(G)/3$. Otherwise, we are guaranteed that $\mathrm{Vol}_G(A^*) \geq 2\,\mathrm{Vol}(G)/3$, and that the conductance of graph $G[A^*]$ is at least ψ.

It now remains to bound the running time of the algorithm. The running time of the algorithm from Corollary 11.6 is bounded by $O\left(\frac{m^{2+O(\epsilon)+o(1)}}{\rho}\right) \leq O\left(\frac{m^{1+o(1)}}{\psi}\right)$, since $\rho = \frac{\psi m}{\tilde{c}^2 \log n}$ and $\epsilon = 1/(\log\log\log m)^{1/25}$. The running time of the algorithm from Theorem 11.9 is $\tilde{O}\left(\frac{|F|}{\psi^2}\right) \leq \tilde{O}\left(\frac{m}{\psi}\right)$. Therefore, the total running time of the algorithm is bounded by $O\left(\frac{m^{1+o(1)}}{\psi}\right)$.

11.4.2 Proof of Theorem 2.9

The proof is very similar to the proof of Theorem 2.8. The main difference is that we set $\rho = m^{1-2\epsilon}$, and that we do not employ Theorem 11.9.

We start by computing a graph $\hat{G}$ exactly as before, and then applying the algorithm from Corollary 11.6 to it, with parameter $\rho = m^{1-2\epsilon}$, and c' a large constant whose value we set later. Let (X^*, Y^*) be the outcome of the algorithm. Recall that both X^* and Y^* are canonical sets of vertices, $|X^*| \geq |Y^*|$, and $|E_{\hat{G}}(X^*, Y^*)| \leq \frac{c'\psi \log^3 n}{\varphi^*} \cdot \hat{n} \leq (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$, since $\varphi^* \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$ and $\epsilon = \frac{1}{(\log\log\log n)^{1/25}}$. We use cut (X^*, Y^*) in $\hat{G}$ in order to define a cut (A, B) in G exactly as before. As before, $\mathrm{Vol}_G(A) = |X^*|$, $\mathrm{Vol}_G(B) = |Y^*|$, and $|E_G(A, B)| = |E_{\hat{G}}(X^*, Y^*)| \leq \psi \cdot (\log n)^{8+o(1)} \cdot \mathrm{Vol}(G)$. As before, if $|Y^*| \geq \hat{n}/3$, then $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \hat{n}/3 = \mathrm{Vol}(G)/3$, and we return cut (A, B) as the outcome of the algorithm.

We assume from now on that $|Y^*| < \hat{n}/3$, and so the algorithm from Corollary 11.6 computed a φ^*-expander graph H with $V(H) = X^*$ and maximum vertex degree $O(\log n)$, together with a set F of at most $O(\rho \log n)$ edges of H, such that there exists an embedding $\mathcal{P}$ of $H \setminus F$ into $\hat{G}[X^*]$ with congestion at most η, where $\eta \leq O\left(\frac{\varphi^*}{c'\psi \log n}\right)$. Notice that in this case, $\mathrm{Vol}_G(A) \geq |X^*| \geq 2\hat{n}/3 = 2\,\mathrm{Vol}(G)/3$ holds.

Consider any partition (Z, Z') of A, with $|E_G(Z, Z')| < \psi \cdot \mathrm{Vol}(G)$, and assume w.l.o.g. that $\mathrm{Vol}_G(Z) \leq \mathrm{Vol}_G(Z')$. We claim that $\mathrm{Vol}_G(Z) < \mathrm{Vol}(G)/100$ holds. Indeed, assume otherwise. Consider a cut $(\hat{Z}, \hat{Z}')$ in graph H, obtained as follows: for every vertex $v_i \in Z$, we add all vertices of V_i to $\hat{Z}$, and for every vertex $v_j \in Z'$, we add all vertices of V_j to $\hat{Z}'$. Clearly, $|\hat{Z}| = \mathrm{Vol}_G(Z) \geq \mathrm{Vol}(G)/100$, and similarly $|\hat{Z}'| > \mathrm{Vol}(G)/100$. Since graph H is a φ^*-expander, $|E_H(\hat{Z}, \hat{Z}')| \geq \varphi^* \cdot \min\left\{|\hat{Z}|, |\hat{Z}'|\right\} \geq$

$\frac{\varphi^* \cdot \mathrm{Vol}(G)}{100}$. Denote $E' = E_H(\hat{Z}, \hat{Z}')$. Recall that $\varphi^* \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$, and so $|E'| \geq \Omega\left(\frac{m}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$. On the other hand, $|F| \leq O(\rho \log n) \leq O(m^{1-2\epsilon} \cdot \log n)$. From our choice of $\epsilon = \frac{1}{(\log\log\log n)^{1/25}}$, we get that $|F| < |E'|/2$, and so $|E' \setminus F| \geq \frac{\varphi^* \cdot \mathrm{Vol}(G)}{200}$. Since there exists an embedding of $H \setminus F$ into $\hat{G}[X^*]$ with congestion at most η, and $\eta \leq O\left(\frac{\varphi^*}{c'\psi \log n}\right)$, we get that:

$$|E_{\hat{G}}(\hat{Z}, \hat{Z}')| \geq \frac{|E'|}{2\eta} \geq \Omega(c'\psi \,\mathrm{Vol}(G) \log n).$$

Since $|E_G(Z, Z')| = |E_{\hat{G}}(\hat{Z}, \hat{Z}')|$, we reach a contradiction to our assumption that $|E_G(Z, Z')| < \psi \cdot \mathrm{Vol}(G)$. We conclude that for every partition (Z, Z') of A with $\mathrm{Vol}_G(Z), \mathrm{Vol}_G(Z') \geq \mathrm{Vol}(G)/100$, $|E_G(Z, Z')| \geq \psi \cdot \mathrm{Vol}(G)$ must hold.

The running time of the algorithm is asymptotically bounded by the running time of the algorithm from Corollary 11.6, which is in turn bounded by $O\left(\frac{m^{2+O(\epsilon)+o(1)}}{\rho}\right) \leq O\left(m^{1+O(\epsilon)+o(1)}\right) \leq O(m^{1+o(1)})$, since $\rho = m^{1-2\epsilon}$ and $\epsilon = 1/(\log\log\log m)^{1/25}$.

11.5 Expander Decomposition – Proof of Theorem 2.10

In this section we prove Theorem 2.10. The proof is essentially identical to the proof of Corollary 6.1 from [CGL+20]. The only difference is that we use our algorithm from Theorem 2.8 instead of the algorithm of [CGL+20].

We maintain a collection $\mathcal{H}$ of disjoint vertex-induced subgraphs of G that we call *clusters*, which is partitioned into two subsets, set $\mathcal{H}^A$ of *active clusters*, and set $\mathcal{H}^I$ of *inactive clusters*. We ensure that for every inactive cluster $H \in \mathcal{H}^I$, the conductance of H is at least ψ. We also maintain a set E' of "deleted" edges, that are not contained in any cluster in $\mathcal{H}$. At the beginning of the algorithm, we let $\mathcal{H} = \mathcal{H}^A = \{G\}$, $\mathcal{H}^I = \emptyset$, and $E' = \emptyset$. The algorithm proceeds as long $\mathcal{H}^A \neq \emptyset$, and consists of iterations. For convenience, we denote $\alpha = (\log n)^{8+o(1)}$, so that the algorithm from Theorem 2.8, when applied to an n-vertex graph G and some parameter ψ', is guaranteed to return a cut (A, B) in G with $|E_G(A, B)| \leq \psi' \cdot \alpha \cdot \mathrm{Vol}(G)$. We set $\psi = \frac{\delta}{c\alpha \cdot \log n}$, where c is the constant from the theorem statement. Clearly, $\psi = \Omega\left(\frac{\delta}{(\log n)^{9+o(1)}}\right)$. Since, from the theorem statement, $\delta > \frac{c \log^{13} m}{m}$ holds, $\psi > \frac{\log^3 m}{m}$ must hold.

In every iteration, we apply the algorithm from Theorem 2.8 to every graph $H \in \mathcal{H}^A$, with the parameter ψ. Consider the cut (A, B) in H that the algorithm returns, with $|E_H(A, B)| \leq \alpha\psi \cdot \mathrm{Vol}(H) \leq \frac{\delta \cdot \mathrm{Vol}(H)}{c \log n}$. We add the edges of $E_H(A, B)$ to set E'. If $\mathrm{Vol}_H(A), \mathrm{Vol}_H(B) \geq \mathrm{Vol}(H)/3$, then we replace H with $H[A]$ and $H[B]$ in $\mathcal{H}$ and in $\mathcal{H}^A$. Otherwise, we are guaranteed that $\mathrm{Vol}_H(A) \geq 2\,\mathrm{Vol}(H)/3$, and graph $H[A]$ has conductance at least ψ. Then we remove H from $\mathcal{H}$ and $\mathcal{H}^A$, add $H[A]$ to $\mathcal{H}$ and $\mathcal{H}^I$, and add $H[B]$ to $\mathcal{H}$ and $\mathcal{H}^A$.

When the algorithm terminates, $\mathcal{H}^A = \emptyset$, and so every graph in $\mathcal{H}$ has conductance at least ψ. Notice that in every iteration, the maximum volume of a graph in $\mathcal{H}^A$ must decrease by a constant factor. Therefore, the number of iterations is bounded by $O(\log m)$. It is easy to verify that the number of edges added to set E' in every iteration is at most $\alpha \cdot \psi \cdot \mathrm{Vol}(G) \leq \frac{\delta \cdot \mathrm{Vol}(G)}{c \log m}$. Therefore, by letting c be a large enough constant, we can ensure that $|E'| \leq \delta \cdot \mathrm{Vol}(G)$. The output of the algorithm is the partition $\Pi = \{V(H) \mid H \in \mathcal{H}\}$ of V. From the above discussion, we obtain a valid (δ, ψ)-expander decomposition, for $\psi = \Omega\left(\frac{\delta}{(\log m)^{9+o(1)}}\right)$.

It remains to analyze the running time of the algorithm. The running time of a single iteration is bounded by $O(m^{1+o(1)}/\psi)$, and, since the number of iterations is $O(\log m)$, the total running time of the algorithm is bounded by $O(m^{1+o(1)}/\psi) \leq O(m^{1+o(1)}/\delta)$.

A Proof of Lemma 4.1

The proof uses a standard ball-growing technique. Let $H = G \setminus E'$. Let S be any set of vertices of H, such that not all vertices of S are isolated in H. We define $L_0 = S$, and, for integers $i > 0$, we define $L_i = B_H(S, i)$. We refer to the sets $L_0, L_1, \ldots$ of vertices as *BFS layers* defined with respect to S. For an index $1 \leq i < \lfloor d/2 \rfloor$, we say that layer L_i is *acceptable* if $|\delta_H(L_i)| < \frac{\varphi}{2} \cdot |E_H(L_i)|$. We use the following simple claim:

Claim A.1 *Let $B = B_H(S, d)$, and assume that $|E_H(B)| < |E(H)|$. Then there is an index $1 \leq i < \lfloor d/2 \rfloor - 1$, such that layer L_i is acceptable.*

Proof: Assume otherwise. Then for all $1 \leq i < \lfloor d/2 \rfloor - 1$, layer L_i is not acceptable, and so $|\delta_H(L_i)| \geq \frac{\varphi}{2} \cdot |E_H(L_i)|$. Therefore, $|E_H(L_{i+1})| \geq \left(1 + \frac{\varphi}{2}\right) |E_H(L_i)|$. Since not all vertices of S are isolated in H, we get that $|E_H(L_1)| \geq 1$. Overall, we get that:

$$|E_H(L_{\lfloor d/2 \rfloor - 1})| \geq \left(1 + \frac{\varphi}{2}\right)^{\lfloor d/2 \rfloor - 2} \geq \left(1 + \frac{\varphi}{2}\right)^{(8 \log m)/\varphi} \geq e^{2 \log m} > m,$$

(we have used the fact that for all $k > 1$, $\left(1 + \frac{1}{k}\right)^{k+1} > e$). This is impossible, so there must be an index $1 \leq i < \lfloor d/2 \rfloor - 1$ for which layer L_i is acceptable. □

We are now ready to complete the proof of Lemma 4.1. We denote the BFS layers in graph H defined with respect to X by $L'_0, L'_1, \ldots$, and BFS layers defined with respect to Y by $L''_0, L''_1, \ldots$. We run two algorithms in parallel. The first algorithm performs a BFS search in H starting from X to compute layers $L'_0, L'_1, \ldots$ one by one. When a layer L'_i is computed, the algorithm checks whether it is acceptable. The second algorithm similarly performs a BFS search starting from Y to compute layers $L''_0, L''_1, \ldots$ one by one, and, when a layer L''_i is computed, the algorithm checks whether it is acceptable. The two algorithms run in parallel, so that at every time point both algorithms have explored the same number of edges of H. The moment one of the two algorithms finds an acceptable layer, we terminate both algorithms.

Assume w.l.o.g. that the first acceptable layer that was computed by either algorithm is L'_i. We then set $X' = L'_i$ and $Y' = V(G) \setminus X'$. Note that, from Claim A.1, $i < d/2$ must hold, so $X' \cap Y = \emptyset$ (as $\mathsf{dist}_H(X, Y) \geq d$). Clearly, $X \subseteq X'$ and $Y \subseteq Y'$. From the definition of an acceptable layer, we are guaranteed that $|E_H(X', Y')| = |\delta_H(X')| < \frac{\varphi}{2} \cdot |E_H(L'_i)| = \frac{\varphi}{2} \cdot |E_H(X')|$. Since we ensured that the number of edges that the two BFS searches explore at every time point is the same, we are guaranteed that $|E_H(X')| \leq |E(Y')|$. The running time of the algorithm is bounded by $O(|E_H(X')| + |\delta_H(X')| + n) \leq O(|E_H(X')| + n) \leq O(|E_G(X')| + n)$.

Lastly, observe that $|E_G(X')| \geq |E_H(X')|$ and $|E_G(Y')| \geq |E_H(Y')|$. Furthermore:

$$|E_G(X', Y')| \leq |E'| + |E_H(X', Y')| \leq \frac{\varphi}{4}|X| + \frac{\varphi}{2} \cdot \min\left\{|E_H(X')|, |E_H(Y')|\right\} \leq \varphi \cdot \min\left\{|E_G(X')|, |E_G(Y')|\right\},$$

since $|X| = |Y|$, and graph G is connected, so $|X| \leq 2 \min\left\{|E_G(X')|, |E_G(Y')|\right\}$.

B Proof of Lemma 11.5

The proof of the lemma is a simple application of the Cut-Matching Game. Let $G' = \hat{G}[V']$, and let $n' = |V(G')|$. If n' is an odd integer, then we add an extra vertex v_0 to G', and connect it with an edge to an arbitrary vertex of G'. We let H be a graph with $V(H) = V(G')$ and $E(H) = \emptyset$.

We will execute the Cut-Matching Game on graph H, while simultaneously computing an embedding of H into G'. Some of the edges of H will be designated as fake edges and added to the set F of fake edges. These edges do not need to be embedded into G'. Initially, $F = \emptyset$.

We perform a number of iterations, that correspond to the Cut-Matching Game. In every iteration i, we will add a matching M_i to graph H, and a set $F_i \subseteq M_i$ of fake edges to set F. We will also implicitly maintain embedding $\mathcal{P}_i$ of the set $M_i \setminus F_i$ of edges into G' (in other words, the paths in $\mathcal{P}_i$ are not computed explicitly, but are only guaranteed to exist). We will ensure that the number of iterations is bounded by $O(\log n') \leq O(\log n)$, so the maximum vertex degree in H is always bounded by $\Delta_H \leq O(\log n)$. At the beginning of the algorithm, graph H contains the set $V[G']$ of vertices and no edges. We now describe the execution of the ith iteration.

In order to execute the ith iteration, we apply Algorithm from Theorem 2.6 to the current graph H, with parameter ϵ remaining unchanged. Notice that, since $\epsilon = \frac{1}{(\log\log\log n)^{1/25}}$, and m is large enough, $\epsilon > \frac{8}{(\log m)^{1/25}}$ holds. Since $|V(G')| \geq \frac{2\hat{n}}{3} \geq m$, we are guaranteed that $\epsilon > \frac{2}{(\log n')^{1/25}}$, so that condition of Theorem 2.6 holds.

Assume first that the output of the algorithm from Theorem 2.6 is a cut (A_i, B_i) in H with $|A_i|, |B_i| \geq n'/4$ and $|E_H(A,B)| \leq n'/100$. We treat this partition as the move of the Cut Player. Assume w.l.o.g. that $|A_i| \leq |B_i|$. Next, we compute an arbitrary partition (A_i', B_i') of $V(G')$ with $|A_i'| = |B_i'|$ and $A_i' \subseteq A_i$. We apply Algorithm MatchOrCut from Theorem 11.2 to graph G', the sets A_i', B_i' of vertices, a sparsity parameter $\varphi' = \varphi/c$, where c is a large enough constant, and parameter $z = 8\rho$. Next, we consider two cases. The first case happens if the algorithm returns a cut (X, Y) in G', with $|X|, |Y| \geq z/2 \geq 4\rho$, and $|E_{G'}(X,Y)| \leq \varphi' \cdot \min\{|X|, |Y|\} = \frac{\varphi}{c} \cdot \min\{|X|, |Y|\}$. Once we delete the extra vertex v_0 (if it exists), we obtain a cut (X', Y') in the original graph G', with $|X'|, |Y'| \geq 2\rho$ and $|E_{G'}(X', Y')| \leq \frac{2\varphi}{c} \cdot \min\{|X'|, |Y'|\}$. Next, we apply Algorithm MakeCanonical from Lemma 11.1, to compute a canonical cut (X'', Y'') in G', such that: $|X''| \geq |X'|/2 \geq \rho$, $|Y''| \geq |Y'|/2 \geq \rho$, and:

$$\begin{aligned}
|E_{\hat{G}}(X'', Y'')| &\leq O(|E_{\hat{G}}(X', Y')|) \\
&= O(|E_{G'}(X', Y')|) \\
&\leq O\left(\frac{\varphi}{c} \cdot \min\left\{|X'|, |Y'|\right\}\right) \\
&\leq \varphi \cdot \min\left\{|X''|, |Y''|\right\},
\end{aligned}$$

if c is sufficiently large. We then terminate the algorithm and return the partition (X'', Y'') of V'. From the above discussion, it has all required properties.

Consider now the second case, where the algorithm from Theorem 11.2 computes a matching $M_i' \subseteq A_i' \times B_i'$ with $|M_i'| \geq |A_i'| - z = |A_i'| - 8\rho$, such that there exists a set $\mathcal{P}_i' = \{P(a,b) \mid (a,b) \in M_i'\}$ of paths in G', where for each pair $(a,b) \in M_i'$, path $P(a,b)$ connects a to b, and the paths in $\mathcal{P}_i'$ cause congestion at most $O\left(\frac{\log n}{\varphi}\right)$. We let $A_i'' \subseteq A_i'$, $B_i'' \subseteq B_i'$ be the sets of vertices that do not participate in the matching M_i', and we let F_i be an arbitrary perfect matching between these vertices. Lastly, we set $M_i = M_i' \cup F_i$. We view the matching M_i as the response of the matching player in the Cut-Matching Game. We add the edges of M_i to H, and continue to the next iteration. Notice that $|F_i| \leq 8\rho$.

We perform the iterations as described above, until the algorithm from Theorem 2.6 returns a subset $S \subseteq V(G')$ of at least $|V(G')|/2$ vertices, such that graph $H[S]$ is φ^*-expander, for $\varphi^* \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \Delta_H^3 \cdot \log^2 n}\right) \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$. Recall that Theorem 3.6 guarantees that this must happen after at most $O(\log n)$ iterations. We then perform one last iteration, whose index we denote by q.

We let $B_q = S$ and $A_q = V(G) \setminus S$, and apply Algorithm MatchOrCut from Theorem 11.2 to the sets A_q, B_q of vertices, a sparsity parameter $\varphi' = \varphi/c$ and parameter $z = 8\rho$. As before, we consider two cases. The first case happens 'if the algorithm returns a cut (X, Y) in G, with $|X|, |Y| \geq z/2 \geq 4\rho$ and $|E_{G'}(X, Y)| \leq \varphi' \cdot \min\{|X|, |Y|\}$. In this case, we compute a partition (X'', Y'') of $V(G') \setminus \{v_0\}$ exactly as before, so that both X'', Y'' are canonical sets of vertices of cardinality at least ρ each, and $|E_{\hat{G}}(X'', Y'')| \leq \varphi \cdot \min\{|X''|, |Y''|\}$. We return the cut (X'', Y'') and terminate the algorithm. In the second case, the algorithm from Theorem 11.2 computes a matching $M'_q \subseteq A'_q \times B'_q$ with $|M'_q| \geq |A_q| - z = |A_q| - 8\rho$, such that there exists a set $\mathcal{P}'_q = \left\{P(a, b) \mid (a, b) \in M'_q\right\}$ of paths in G', where for each pair $(a, b) \in M'_q$, path $P(a, b)$ connects a to b, and the paths in $\mathcal{P}'_q$ cause congestion at most $O\left(\frac{\log n}{\varphi}\right)$. As before, we let $A'_q \subseteq A_q$, $B'_q \subseteq B_q$ be the sets of vertices that do not participate in the matching M'_q, and we let F_q be an arbitrary matching that connects every vertex of A'_q to a distinct vertex of B'_q (such a matching must exist since $|A_q| \leq |B_q|$). We then set $M_q = M'_q \cup M''_q$, and we add the edges of M_q to graph H.

From now on we assume that the algorithm never terminated with a partition (X'', Y'') of $V(G') \setminus \{v_0\}$, where both X'', Y'' are canonical sets of vertices of cardinality at least ρ each, and $|E_{\hat{G}}(X'', Y'')| \leq \varphi \cdot \min\{|X''|, |Y''|\}$. Note that, from Observation 3.2, the final graph H is a $\varphi^*/2$-expander, for $\varphi^* \geq \Omega\left(\frac{1}{2^{O(1/\epsilon^6)} \cdot \log^5 n}\right)$. Let $F = \bigcup_i F_i$. Since, for all i, $|F_i| \leq 8\rho$, and since, from Theorem 3.6, the number of iterations is bounded by $O(\log n)$, we get that $|F| \leq O(\rho \log n)$. Lastly, consider the set $\mathcal{P} = \bigcup_i \mathcal{P}_i$ of paths in graph G'. It is immediately to verify that the paths in $\mathcal{P}$ embed graph $H \setminus F$ into G'. Since every set $\mathcal{P}_i$ of paths causes congestion at most $O\left(\frac{\log n}{\varphi}\right)$, the paths in $\mathcal{P}$ cause congestion at most $O\left(\frac{\log^2 n}{\varphi}\right)$ in G'.

One remaining subtlety is that graph H, as well as current graph G' may contain the extra vertex v_0, that needs to be removed from both graphs. Recall that the degree of v_0 in graph H is at most $O(\log n)$. Let $u_1, \ldots, u_r$ denote the neighbor vertices of v_0 in H. Let H' be obtained from graph H by deleting vertex v_0 from it, and adding, for every pair $u_j, u_{j'}$ of neighbor vertices of v_0, and edge $(u_j, u_{j'})$ connecting them. Each such new edge is added to the set F of fake edges. It is easy to verify that H' remains a $\varphi^*/2$-expander. Since $\rho \geq \log n$, while the degree of v_0 in H is at most $O(\log n)$, $|F| \leq O(\rho \log n)$ continues to hold, and all vertex degrees in H' are at most $O(\log n)$. Since vertex v_0 has degree 1 in G', we can assume that it does not lie on any path in $\{P(e) \mid e \in E(H') \setminus F\}$, and so v_0 can be safely deleted from G' as well. The output of the algorithm in this case is graph H' and set F of its edges.

Lastly, we bound the running time of the algorithm. The algorithm consists of $O(\log n)$ iterations. Every iteration employs the algorithm from Theorem 2.6, whose running time is $O\left(|E(H)|^{1+O(\epsilon)} \cdot \Delta_H^7\right) \leq O\left(n^{1+O(\epsilon)}\right)$, since $\Delta_H \leq O(\log n)$, and $\log^8 n < n^{4\epsilon}$ (since, as we have observed, $n^{2\epsilon} \geq m^{\epsilon} > \log^4 n$). Additionally, in every iteration we use Algorithm MatchOrCut from Theorem 11.2, whose running time is $O\left(m^{1+o(1)}\right)$, and Algorithm MakeCanonical from Lemma 11.1, whose running time is $O(m)$. Therefore, the total running time is $O\left(m^{1+O(\epsilon)+o(1)}\right)$.

References

[ACL07] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 4(1):35–64, 2007.

[ADK22] Daniel Agassy, Dani Dorfman, and Haim Kaplan. Expander decomposition with fewer inter-cluster edges using a spectral cut player. *arXiv preprint arXiv:2205.10301*, 2022.

[Alo86] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), 2009.

[CGL+20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167. IEEE, 2020. Full version at arXiv:1910.08025.

[Che18] Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 170–181. IEEE, 2018.

[Chu21] Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 626–639, 2021. Full version at arXiv:2109.05621.

[CK19] Julia Chuzhoy and Sanjeev Khanna. A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 389–400, 2019.

[CS19] Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019.*, pages 66–73, 2019.

[CS21] Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2478–2496. SIAM, 2021.

[DHZ00] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5):1740–1759, 2000.

[Din06] Yefim Dinitz. Dinitz' algorithm: The original version and Even's version. In *Theoretical computer science*, pages 218–240. Springer, 2006.

[ES81] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

[Fle00] Lisa Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

[GG81] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. Syst. Sci.*, 22(3):407–420, 1981. announced at FOCS'79.

[GK98] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 300–309, 1998.

[GLN+19] Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Deterministic graph cuts in subquadratic time: Sparse, balanced, and k-vertex. *arXiv preprint arXiv:1910.07950*, 2019.

[GVY95] N. Garg, V.V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)-cut theorems and their applications. *SIAM Journal on Computing*, 25:235–251, 1995.

[HK95] Monika Rauch Henzinger and Valerie King. Fully dynamic biconnectivity and transitive closure. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 664–672. IEEE, 1995.

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015.

[Kar08] George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Trans. Algorithms*, 4(1):13:1–13:17, 2008.

[KKOV07] Rohit Khandekar, Subhash Khot, Lorenzo Orecchia, and Nisheeth K Vishnoi. On a cut-matching game for the sparsest cut problem. *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 6(7):12, 2007.

[KMP12] Jonathan A. Kelner, Gary L. Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1–18, 2012.

[KRV09] Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)*, 56(4):19, 2009.

[LR99] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46:787–832, 1999.

[Mad10] Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 121–130, 2010.

[Mar73] G. A. Margulis. Explicit construction of concentrators. *Problemy Peredafi Iqfiwmacii*, 9(4):71–80, 1973. (English translation in Problems Inform. Transmission (1975)).

[NS17] Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2-\epsilon})$-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1122–1129, 2017.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90. ACM, 2004.

[SW19] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2616–2635, 2019.

[Wul17] Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1130–1143, 2017.