Maximum Bipartite Matching in $n^{2+o(1)}$ Time via a Combinatorial Algorithm

Julia Chuzhoy[*] Sanjeev Khanna [†]

May 31, 2024

Abstract

Maximum bipartite matching (MBM) is a fundamental problem in combinatorial optimization with a long and rich history. A classic result of Hopcroft and Karp [HK73] provides an $O(m\sqrt{n})$-time algorithm for the problem, where n and m are the number of vertices and edges in the input graph, respectively. For dense graphs, an approach based on fast matrix multiplication achieves a running time of $O(n^{2.371})$ [IM81, MS04]. For several decades, these results represented state-of-the-art algorithms, until, in 2013, Madry introduced a powerful new approach for solving MBM using continuous optimization techniques. This line of research, that builds on continuous techniques based on interior-point methods, led to several spectacular results, culminating in a breakthrough $m^{1+o(1)}$-time algorithm for min-cost flow, that implies an $m^{1+o(1)}$-time algorithm for MBM as well.

These striking advances naturally raise the question of whether combinatorial algorithms can match the performance of the algorithms that are based on continuous techniques for MBM. One reason to explore combinatorial algorithms is that they are often more transparent than their continuous counterparts, and that the tools and techniques developed for such algorithms may be useful in other settings, including, for example, developing faster algorithms for maximum matching in general graphs. A recent work of Chuzhoy and Khanna [CK24] made progress on this question by giving a combinatorial $\tilde{O}(m^{1/3}n^{5/3})$-time algorithm for MBM, thus outperforming both the Hopcroft-Karp algorithm and matrix multiplication based approaches, on sufficiently dense graphs. Still, a large gap remains between the running time of their algorithm and the almost linear-time achievable by algorithms based on continuous techniques. In this work, we take another step towards narrowing this gap, and present a randomized $n^{2+o(1)}$-time combinatorial algorithm for MBM. Thus in dense graphs, our algorithm essentially matches the performance of algorithms that are based on continuous methods.

Similar to the classical algorithms for MBM and the approach of [CK24], our algorithm is based on iterative augmentation of a current matching using augmenting paths in the corresponding (directed) residual flow network. Our main contribution is a recursive algorithm that exploits the special structure of the resulting flow problem to recover an $\Omega(1/\log^2 n)$-fraction of the remaining augmentations in $n^{2+o(1)}$ time.

Finally, we obtain a randomized $n^{2+o(1)}$-time algorithm for maximum vertex-capacitated s-t flow in directed graphs when all vertex capacities are identical, using a standard reduction from this problem to MBM.

[*]Toyota Technological Institute at Chicago. Email: `cjulia@ttic.edu`. Supported in part by NSF grant CCF-2006464 and NSF HDR TRIPODS award 2216899.

[†]University of Pennsylvania, Philadelphia, PA. Email: `sanjeev@cis.upenn.edu`. Supported in part by NSF awards CCF-1934876 and CCF-2008305.

Contents

1 Introduction

We consider the classical Maximum Bipartite Matching problem, where the goal is to compute a maximum-size matching in the given input bipartite graph G. Throughout, we denote the number of vertices and the number of edges in G by n and m, respectively. Maximum Bipartite Matching is one of the most central and extensively studied problems in computer science and related disciplines, with connections to many other fundamental graph optimization problems.

It is well known that Maximum Bipartite Matching can be reduced to computing a maximum s-t flow in a directed flow network with unit edge capacities. The Ford-Fulkerson algorithm [FF56] for maximum s-t flow then immediately implies an $O(mn)$-time algorithm for Maximum Bipartite Matching. The algorithm is conceptually simple, and maintains a matching M, starting with $M = \emptyset$. As long as M is not optimal, we can augment it by computing an s-t path in the resulting residual flow network. A celebrated work of Hopcroft and Karp [HK73] provides a significantly more efficient $O(m\sqrt{n})$-time implementation of this idea by iteratively computing a maximal collection of internally disjoint augmenting s-t paths of shortest possible length in the residual flow network. This result remained the fastest known algorithm for several decades, except for the special case of very dense graphs, where fast matrix multiplication techniques were shown to yield an $O(n^{\omega})$-time algorithm [IM81, MS04]. Starting in 2008, a new paradigm emerged, namely, the use of continuous techniques as a method for obtaining fast algorithms for various flow problems, that ultimately revolutionized the field. As a first illustration of this paradigm, the work of Daitch and Spielman [DS08], building on the breakthrough result of Spielman and Teng [ST04] for efficiently solving Laplacian systems, gave an $\widetilde{O}(m^{3/2})$-time algorithm for directed maximum s-t flow. Later, Madry [Mad13] used this paradigm to design an algorithm for directed maximum s-t flow with $\tilde{O}(m^{10/7})$ running time, obtaining the first substantial improvement over the algorithm of Hopcroft and Karp for Maximum Bipartite Matching in sparse graphs. A sequence of remarkable developments [Mad16, LS19, CMSV17, LS20, AMV20, vdBLN+20] recently culminated in a deterministic $m^{1+o(1)}$-time algorithm for directed maximum s-t flow [CKL+22, vdBCK+23], thereby providing an almost linear-time algorithm for Maximum Bipartite Matching. In all these recent algorithms, the directed flow problem is cast as a linear program, which is then solved via interior-point methods (IPM). In every iteration of the IPM, one needs to either solve a Laplacian system, or another efficiently solvable problem on undirected graphs, such as min-ratio cycle in [CKL+22]. This approach is further combined with dynamic graph data structures to make it even more efficient.

In view of this recent history, it is natural to ask whether combinatorial techniques can be used to design algorithms for Maximum Bipartite Matching (and also other flow-like problems), whose performance matches that of algorithms that are based on continuous methods. There are several reasons to focus on combinatorial techniques. First, they tend to be more transparent than their continuous counterparts. Second, it is likely that tools and techniques that are developed in order to design a combinatorial algorithm for as fundamental a problem as Maximum Bipartite Matching will prove useful in other applications. Lastly, while continuous techniques led to an $m^{1+o(1)}$-time algorithm for Maximum Bipartite Matching, the landscape of fast algorithms for the Maximum Matching problem in general graphs did not benefit from these developments. In dense graphs, a fast-matrix multiplication based approach gives $O(n^{2.371})$-time algorithm for Maximum Matching in general graphs [IM81, MS04]. More interestingly, in sparse to moderately dense graphs, the best known runtime still stands on $\widetilde{O}(m\sqrt{n})$ [MV80, Vaz94, GK04, Gab17] and utilizes an augmenting-paths based approach, similar to that used in combinatorial algorithms for Maximum Bipartite Matching.

In a very recent work, Chuzhoy and Khanna [CK24] made progress on narrowing the striking gap between the performance of combinatorial and IPM-based approaches for Maximum Bipartite Matching, by providing a combinatorial deterministic $\tilde{O}(m^{1/3}n^{5/3})$-time algorithm, thus outperforming both the Hopcroft-Karp algorithm, and the matrix multiplication based approaches on sufficiently dense graphs.

Still, a large gap remains between the performance of the best combinatorial algorithms and the almost linear-time achievable by algorithms based on continuous techniques. In particular, on dense graphs, the performance gap incurred by the current best combinatorial algorithm is $\Omega(n^{1/3})$. In this work, we take another step towards narrowing this performance gap, and essentially eliminate it in dense graphs. Our main result is summarized in the following theorem.

Theorem 1.1 *There is a randomized combinatorial algorithm for the* Maximum Bipartite Matching *problem, that, given an n-vertex bipartite graph G, outputs a maximum matching M in G with probability at least $1 - 1/\operatorname{poly}(n)$. The running time of the algorithm is $O\left(n^2 \cdot 2^{O(\sqrt{\log n} \cdot \log\log n)}\right)$.*

Our algorithm outperforms the Hopcroft-Karp algorithm on graphs with $\omega(n^{1.5})$ edges, and in dense graphs, it essentially matches the performance of algorithms based on continuous techniques. Furthermore, in almost all edge density regimes, this algorithm outperforms the runtime achieved in [CK24].

Using a standard reduction from vertex-capacitated flow in directed graphs to Maximum Bipartite Matching (see Theorem 16.12 in [Sch03], for instance), we also obtain a combinatorial algorithm with similar running time for maximum vertex-capacitated flow when all vertex capacities are identical.

Corollary 1.2 *There is a randomized combinatorial algorithm for the directed maximum s-t flow problem with uniform vertex capacities, that given an n-vertex directed graph G, outputs a maximum s-t flow with probability at least $1-1/\operatorname{poly}(n)$. The running time of the algorithm is $O\left(n^2 \cdot 2^{O(\sqrt{\log n} \cdot \log\log n)}\right)$.*

Similarly to the classical algorithms for Maximum Bipartite Matching, our approach for proving Theorem 1.1 is based on iteratively augmenting a current matching using augmenting paths in the residual flow network. We employ the multiplicative weights update (MWU) framework, that effectively reduces the underlying flow problem to decremental single-source shortest paths (SSSP) in directed graphs, a connection first observed by Madry [Mad10] and also used in [CK24]. As observed in [CK24], this reduction results in a *special case* of decremental SSSP that appears significantly easier than general decremental directed SSSP. Our main contribution is a recursive algorithm that exploits the special structure of the resulting flow problem to recover an $\Omega(1/\log^2 n)$-fraction of the remaining augmentations in $n^{2+o(1)}$ time. We abstract this task as a problem called RouteAndCut, where the input is a directed graph G, two disjoint sets A, B of its vertices with $|A| \leq |B|$, and two additional parameters $1 \leq \eta \leq \Delta$. The goal is to either compute a collection $\mathcal{P}$ of at least $\Omega(\Delta/\operatorname{poly}\log n)$ paths that connects distinct vertices of A to distinct vertices of B with vertex-congestion at most η; or to output a cut that approximately certifies infeasibility of the desired routing. Our main result is a randomized algorithm for RouteAndCut, whose running time is bounded by $n^{1+o(1)} \cdot (n - |B|)$. It is worth highlighting that when $|B|$ is sufficiently large, this running time may be much smaller than $|E(G)|$. This performance gain for large sets B serves as a crucial building block for our $n^{2+o(1)}$-time algorithm.

As in the work of [CK24], the task of efficiently solving RouteAndCut in turn relies on an efficient algorithm for maintaining an expander in a dynamically changing graph, a problem that we refer to as MaintainCluster. One key contribution of our work is the introduction of a *parameterized version* of both these problems that allows us to use a bootstrapping approach in the design of our algorithm, where we exploit efficient algorithms for one problem to obtain efficient algorithms for the other problem and vice versa. Another key technical contribution is an efficient algorithm for a new problem that we introduce, called ConnectToCenters, that plays a central role in efficiently maintaining short paths from all vertices of a given graph G to a pre-specified collection of "center" vertices, even as G undergoes online updates. This problem may be viewed as a representative abstraction of a common paradigm used in many graph algorithms, in which an expander is embedded into the input graph, and all graph vertices are then routed to the vertices of the expander. As such, our algorithm for this problem may prove useful in other applications. Finally, another insight utilized by our algorithm is an explicit recognition of the fact that each iteration of the MWU algorithm leads to very specific kind of updates in the underlying SSSP instance, namely, well-behaved increases in the lengths of some edges. While these length increases can easily be simulated as edge deletions, a black-box simulation as an instance of decremental SSSP gives away some of the inherent algorithmic advantages offered by these special kind of updates that our algorithm exploits. We give a detailed overview of our algorithm and its comparison to the algorithm of [CK24] in the next subsection.

We conclude by noting that in addition to the conceptual simplicity of a combinatorial augmenting path based approach to solve Maximum Bipartite Matching, the techniques developed here for speeding up augmentations may also prove useful in obtaining faster algorithms for Maximum Matching in general graphs. We also believe that some of the technical tools that we introduce, such as an efficient algorithm for the ConnectToCenters problem that we describe in more detail below, are of independent interest.

1.1 Our Techniques

Our algorithm builds on and extends the techniques of [CK24], which, in turn, build on the algorithm of [BGS20] for the directed decremental Single-Source Shortest Path (SSSP) problem. We start with a high-level overview of the algorithm of [CK24], and then provide the description of our improved algorithm.

It is well known that the Maximum Bipartite Matching problem in a graph G can be equivalently cast as the problem of computing an integral maximum s-t flow in a corresponding directed flow network G' with unit edge capacities. We can view any given matching M in G as defining an s-t flow f in G' of value $|M|$. We let $H = G'_f$ be the corresponding residual flow network, that we refer to as the *residual flow network corresponding to matching M*. We note that the residual flow network H has a special structure: namely, each vertex in H has in-degree 1 or out-degree 1. Therefore, if $\mathcal{P}$ is a collection of paths in H causing edge-congestion at most η, then the paths in $\mathcal{P}$ cause vertex-congestion at most η and vice versa. For all problems that we define below, we assume that their input graph also has this special structure. For convenience, we will focus on edge-congestion, and on edge-based cuts in such graphs. We also note that any directed graph can be converted into a graph with this special structure by replacing every vertex v with a pair v^+, v^- of new vertices, such that all edges that enter v become incident to v^-, and all edges leaving v become incident to v^+, and inserting the edge (v^-, v^+) into the graph.

The residual network H corresponding to a matching M contains $\Delta = \mathsf{OPT} - |M|$ edge-disjoint s-t paths, where OPT is the value of the maximum bipartite matching. Suppose now that we can design an algorithm that computes a collection $\mathcal{P}$ of $\Omega(\Delta/\operatorname{poly}\log n)$ s-t paths in H, that cause $O(\operatorname{poly}\log n)$ edge-congestion. Using standard methods, we can then efficiently recover a collection $\mathcal{P}'$ of $\Omega(\Delta/\operatorname{poly}\log n)$ edge-disjoint s-t paths in H, which can in turn be used in order to augment the current matching M, thereby obtaining a new matching M' of cardinality $|M| + \Omega(\Delta/\operatorname{poly}\log n)$. In other words, $\mathsf{OPT} - |M'| \leq (\mathsf{OPT} - |M|) \cdot (1 - 1/\operatorname{poly}\log n)$, so the gap between the optimal solution value and the size of the matching the algorithm maintains reduces by at least factor $(1-1/\operatorname{poly}\log n)$. It is then easy to verify that, after $O(\operatorname{poly}\log n)$ such iterations, the algorithm obtains an optimal matching. This is precisely the high-level approach that was used by [CK24], and we follow it in this work as well. In order to obtain an algorithm for Maximum Bipartite Matching, it is now sufficient to design a procedure that, given a residual flow network H corresponding to the current matching M, efficiently computes the set $\mathcal{P}$ of s-t paths in H with the above properties.

For technical reasons that will become clear later, we define a slightly more general problem, that we call RouteAndCut. In this problem, the input is a directed graph H, two disjoint sets A, B of its vertices with $|A| \leq |B|$, and two parameters $1 \leq \Delta \leq \min\{|A|, |B|\}$ and $1 \leq \eta \leq \Delta$. The goal is to either compute a collection $\mathcal{Q}$ of $\Omega(\Delta/\operatorname{poly}\log n)$ paths, each of which connects a distinct vertex of A to a distinct vertex of B, such that the paths in $\mathcal{Q}$ cause congestion $\tilde{O}(\eta)$; or to compute a cut (X, Y) in H with $|E_H(X, Y)| \ll \Delta/\eta$, with X containing a large fraction of vertices of A, and Y containing a large fraction of the vertices of B. While [CK24] do not explicitly define and solve this problem, their algorithm can be adapted to solve a special case of RouteAndCut, where $\eta \leq O(\operatorname{poly}\log n)$. So for brevity, we will say that the algorithm of [CK24] solves this special case of RouteAndCut. Clearly, an algorithm for the RouteAndCut problem can be used in order to compute a collection $\mathcal{P}$ of paths in the residual flow network H corresponding to the current matching M, with the desired properties that we described above.

The RouteAndCut problem falls into the broader and the extensively studied class of graph routing and flow problems. One standard approach for obtaining fast algorithms for such problems, due to [GK98, Fle00, AAP93], is via the Multiplicative Weight Update (MWU) method. It was further observed by Madry [Mad10] that this approach can be viewed as reducing a given flow problem to a variant of decremental SSSP or APSP. In our case, the reduction is to decremental SSSP in directed graphs. While strong lower bounds are known for exact algorithms for decremental SSSP and APSP (see, e.g. [DHZ00, RZ11, HKNS15, ABKZ22, ABF23]), we can exploit the special properties of the SSSP instances that arise from the RouteAndCut problem in order to obtain faster algorithms, an approach that was also used by [CK24].

We note that [BGWN20] provided a $(1+\epsilon)$-approximation algorithm for directed decremental SSSP with total update time $\widetilde{O}(n^2/\epsilon)$, assuming all edge lengths are poly-bounded. Unfortunately, their algorithm can only withstand an *oblivious adversary*; in other words, the sequence of edge-deletions from the input graph G must be fixed in advance and may not depend on the algorithm's responses to queries. Instances of decremental SSSP arising from the MWU framework crucially require algorithms that can withstand an *adaptive adversary*, where the choice of the edge to be deleted in every update may depend on the algorithm's past behavior. A recent work of [BGS20] provided a $(1+\epsilon)$-approximation algorithm for the directed decremental SSSP problem with an adaptive adversary, that achieves total update time $O\left(\frac{n^{8/3+o(1)}}{\epsilon}\right)$ (assuming that all edge lengths are poly-bounded). While this total update time is too high for speeding up algorithms for Maximum Bipartite Matching, their approach was adapted by [CK24] to handle the specific instances of SSSP that they obtain, leading to faster algorithms for Maximum Bipartite Matching. Specifically, one of the key observations of [CK24] is that the SSSP instances that arise from applying the MWU method to the Maximum Bipartite Matching problem have the property that all queries are between a fixed pair (s, t) of vertices, and a rather large approximation factor is acceptable. Moreover, by slightly modifying the standard MWU framework, they ensure that it is sufficient that the algorithm for the SSSP problem only responds to shortest-path queries as long as the current graph H contains a collection of least $\Omega(\Delta/\operatorname{poly}\log n)$ disjoint and short s-t paths, where Δ is the target number of augmenting paths that the algorithm aims to produce. We also follow their approach, and reduce the RouteAndCut problem, via a slightly modified MWU method, to a special case of directed decremental SSSP, that we refer to as decremental s-t-SP, that has all of the above properties. We note that s-t-SP is somewhat more general than the special case of the SSSP problem that was considered in [CK24], since they only provide an algorithm for the special case of RouteAndCut where $\eta \leq O(\operatorname{poly}\log n)$, while we need an algorithm that works for a wider range of values of parameter η. For now we focus on the description of their algorithm, and we assume for simplicity that $\eta = 1$ in this discussion.

The algorithm of [CK24] for a special case of the decremental s-t-SP problem follows the high-level approach of [BGS20], that consists of two parts. First, they maintain a partition $\mathcal{X}$ of graph $H \setminus \{s, t\}$ into a collection of *expander-like graphs*; we abstract the problem of maintaining each such graph, that we call MaintainCluster problem, below. Intuitively, the MaintainCluster subroutine is given as input a vertex-induced subgraph H' of H, and a distance parameter d, with H' undergoing an online sequence of edge deletions. It needs to efficiently support short-path queries in H': given a pair $x, y \in V(H')$ of vertices, return an x-y path of length at most d in H'. However, it may, at any time, produce a cut (A, B) in H' of sparsity at most $O\left(\frac{\text{poly}\log n}{d}\right)$, after which the vertices on one side of the cut are deleted from H', and the algorithm needs to continue with the resulting graph. The second main ingredient in the algorithm of [CK24] is the Approximate Topological Order ($\mathcal{ATO}$) framework of [BGWN20] (which is in turn based on the works of [GWN20] and [Ber17]), combined with the algorithm of [BGWN20] for decremental SSSP on "almost" DAG's. The latter algorithm is applied to the graph $\hat{H}$, that is obtained from H by contracting every almost-expander $X \in \mathcal{X}$ into a single vertex. We now discuss each of these components in turn, starting with the $\mathcal{ATO}$ framework.

The $\mathcal{ATO}$ framework. The Approximate Topological Order ($\mathcal{ATO}$) framework of [BGWN20][1] is a central component in the algorithms of [BGS20, CK24], as well as our algorithm. The framework was first introduced as the means of reducing the decremental SSSP problem on general directed graphs to decremental SSSP on "almost DAG's".

An $\mathcal{ATO}$ data structure in a dynamic graph H must maintain a *partition* $\mathcal{X}$ of the vertices of H into subsets. We refer to the sets $X \in \mathcal{X}$ as *clusters*, and to $\mathcal{X}$ as a *clustering*. The only allowed changes to the clustering $\mathcal{X}$ are *cluster splittings*: given an existing cluster $X \in \mathcal{X}$ and a subset $X' \subseteq X$ of its vertices, delete the vertices of X' from X, and add X' as a new cluster to $\mathcal{X}$. We assume further that the input graph H contains two special vertices s and t, and that clusters $S = \{s\}$ and $T = \{t\}$ always lie in $\mathcal{X}$. In addition to maintaining the clustering $\mathcal{X}$, the $\mathcal{ATO}$ must maintain an *ordering* σ of its clusters. Assume that $\mathcal{X} = \{X_1, \ldots, X_r\}$, and that the clusters are indexed according to the ordering σ. Assume further that a cluster X_i undergoes splitting, with the new cluster $X'_i \subseteq X_i$ inserted into $\mathcal{X}$. Then the ordering σ must evolve in a *consistent* manner, that is, the new ordering must be either $(X_1, \ldots, X_{i-1}, X'_i, X_i \setminus X'_i, X_{i+1}, \ldots, X_r)$, or $(X_1, \ldots, X_{i-1}, X_i \setminus X'_i, X'_i, X_{i+1}, \ldots, X_r)$. Consider now some edge $e = (x, y)$ of H, and assume that $x \in X_i$ and $y \in X_j$. If X_i appears before X_j in the ordering σ, then we say that e is a *left-to-right* edge; if $i = j$, we say that it is a *neutral* edge; and otherwise we say that it is a *right-to-left* edge. If e is a right-to-left edge, then we define its *span*: $\text{span}(e) = \sum_{i'=j}^{i} |X_{i'}|$ (we assume here that the sets in $\mathcal{X}$ are indexed according to the ordering σ). Let $\hat{H}$ be the contracted graph corresponding to H: that is, $\hat{H}$ is obtained from H by contracting each of the clusters $X \in \mathcal{X}$ into a vertex v_X. For simplicity, we will refer to the vertices v_S and v_T as s and t, respectively. Intuitively, if we could maintain the $\mathcal{ATO}$ without introducing any right-to-left edges, then the corresponding contracted graph $\hat{H}$ is a DAG, and the ordering σ associated with the $\mathcal{ATO}$ naturally defines a topological ordering of the vertices of $\hat{H}$. We could then use the algorithm of [BGWN20] for decremental SSSP in DAG's, that builds on the work of [Ber17, GWN20], in order to support approximate shortest path queries in $\hat{H}$ between s and other vertices of $\hat{H}$, with total update time $\widetilde{O}(n^2)$. But in order to be able to support shortest s-t path queries in the original graph H, we need to ensure that the diameters of the subgraphs $H[X]$ corresponding to the clusters $X \in \mathcal{X}$ are sufficiently small, and that we can support approximate shortest-path queries between *arbitrary pairs* of vertices within each such graph efficiently.

Towards this end, it was observed by [BGWN20] that the algorithm for decremental SSSP in DAG's can be further extended to "almost DAG's": suppose G is a directed graph, and let ρ be a fixed ordering of its vertices. Assume that $V(G) = \{v_1, \ldots, v_n\}$, where the vertices are indexed according to the ordering ρ. If $e = (v_i, v_j)$ is an edge with $i > j$, then we say that e is a *right-to-left* edge of G, and

[1]According to [BGWN20], while the $\mathcal{ATO}$ technique was first explicitly defined in [BGWN20], the work of [GWN20] can be viewed as (implicitly) using this technique.

we say that its *width* is $i-j$. It was shown in [BGWN20] that the algorithm for decremental SSSP on DAG's could be extended to such graphs G, provided that the total width of all right-to-left edges is relatively small. Specifically, the running time of their algorithm becomes roughly $\tilde{O}(n^2+\Gamma\cdot n)$, where Γ is the total width of the right-to-left edges.

Assume now that the algorithm for SSSP on the almost-DAG graph G only needs to respond to approximate shortest-path queries between a specific fixed pair s,t of vertices, and assume, moreover, that it is only needed to support such queries as long as G contains $\Omega(\Delta)$ short edge-disjoint s-t paths. It was observed in [CK24] that, in such a case, the running time of the algorithm of [BGWN20] improves to roughly $\tilde{O}(n^2+\Gamma\cdot n/\Delta)$. This observation was one of the key insights that allowed them to obtain a faster running time for the special case of the s-t-SP problem, and for Maximum Bipartite Matching.

We now provide additional relevant details of the algorithm of [CK24]. Like in [BGS20], the MaintainCluster problem is exploited in order to maintain an $\mathcal{ATO}$ of the input graph H. Initially, the clustering $\mathcal{X}$ contains three clusters: $S=\{s\}$, $T=\{t\}$, and $U=V(H)\setminus\{s,t\}$. The algorithm for the MaintainCluster problem is then initialized on graph $H[U]$, with an appropriately chosen distance parameter d_U. In general, whenever a new cluster X joins $\mathcal{X}$, the algorithm for the MaintainCluster problem is initialized on $H[X]$. Whenever that algorithm produces a sparse cut (A,B) in X, we select a subset $Z\in\{A,B\}$ of vertices to be deleted from X, update X by deleting these vertices, and add Z as a new cluster to $\mathcal{X}$, after which the algorithm for the MaintainCluster problem is initialized on $H[Z]$. The key idea is that, since the cuts produced by the algorithm for the MaintainCluster problem are sparse, we can ensure that the total span of all right-to-left edges is sufficiently small. If we then consider the contracted graph $\hat{H}$, this, in turn, ensures that the total width of all right-to-left edges in $\hat{H}$ is low. We can now apply the algorithm of [BGWN20] for decremental SSSP on almost-DAG's to support approximate shortest s-t path queries in $\hat{H}$, while exploiting the fact that such queries only need to be supported as long as $\hat{H}$ contains a large number of short edge-disjoint s-t paths, in order to speed it up. For every cluster $X\in\mathcal{X}$, we can then use the algorithm for the MaintainCluster problem on $H[X]$, in order to respond to approximate shortest-path queries between pairs of vertices in X. Combining these data structures together, we can support approximate shortest s-t path queries in the original graph H, as long as H contains many short edge-disjoint s-t paths. This high-level approach allows one to obtain algorithms for decremental s-t-SP, and for the RouteAndCut problem, from algorithms for the MaintainCluster problem, that we now discuss in more detail.

The MaintainCluster problem. To recap, in the MaintainCluster problem, the input is a graph G that undergoes an online sequence of edge deletions, and a distance parameter d. The goal is to efficiently support short-path queries: given a pair x,y of vertices of G, return a path of length at most d connecting them in G. The algorithm may, however, at any time, produce a cut (A,B) in G of sparsity at most $O\left(\frac{\operatorname{poly}\log n}{d}\right)$, following which, the vertices of one side of the cut are deleted from G. The algorithm is used in order to maintain individual clusters of the $\mathcal{ATO}$. A similar problem was considered by [BGS20], who provide an algorithm with total update time $\widehat{O}(|E(G)|\cdot d^2)$ for it, where the time to respond to each query is roughly proportional to the number of edges on the path that the algorithm returns. In [CK24] this problem was considered in a more relaxed setting, where the number of queries that the algorithm must support is bounded by a given parameter Δ, and the goal is to minimize the total running time of the algorithm, that is, the sum of the total update time, and the time required to respond to all queries. The algorithm of [CK24] for this setting has running time $\widetilde{O}(|E(G)|\cdot d+|V(G)|^2)\cdot\max\left\{1,\frac{\Delta\cdot d^2}{|V(G)|}\right\}$, which, for the specific parameters that they employ, becomes $\widetilde{O}(|E(G)|\cdot d+|V(G)|^2)$. In order to obtain our improved algorithm for Maximum Bipartite Matching, we need to generalize this result so that it works for a wider range of parameters, achieving running time $|V(G)|^{2+o(1)}$.

The algorithm of [CK24] follows a rather standard approach. First, they use the Cut-Matching game in order to compute a large expander graph $\hat{G}$, and to embed it into G via short paths that cause low congestion. The algorithm for the Cut Player is implemented in a rather straightforward manner, since they can afford a running time that is as high as $\Theta(|V(\hat{G})|^2)$. The algorithm for the Matching Player essentially needs to solve an instance of the RouteAndCut problem. Using the MWU approach as before, it can be reduced to solving an instance of directed decremental SSSP. The algorithm of [CK24] then uses the standard Even-Shiloach tree data structure in order to solve the latter problem. In addition to the expander $\hat{G}$ and its embedding into G, the algorithm of [CK24] maintains two additional Even-Shilach trees in G. Both trees are rooted in the vertices of $\hat{G}$, and have depth roughly d. One of the trees has all its edges directed away from the root, and the other has all of its edges directed towards the root. In order to respond to a query between a pair x, y of vertices of G, the two Even-Shiloach trees are used to compute a short path connecting x to some vertex $x' \in V(\hat{G})$, and a short path connecting some vertex $y' \in V(\hat{G})$ to y. A simple BFS search in the expander $\hat{G}$ then yields a short path connecting x' to y', which can be turned into an x'-y' path in G by exploiting the embedding of $\hat{G}$ into G.

We now describe several sources of inefficiency of the algorithm of [CK24], and then describe our approach to overcoming them. First, both the algorithms of [CK24] and [BGS20] for the MaintainCluster problem are only designed for graphs with unit edge-length. However, both of these works solve (a variant of) the SSSP problem in graphs with arbitrary edge lengths. To overcome this difficulty, [CK24] use the same approach as [BGS20]: namely, they choose a threshold τ, and initially delete all edges whose length is greater than τ (called *long edges*) from the input graph H. The lengths of the remaining edges (called *short edges*) are set to 1 for the sake of maintaining the $\mathcal{ATO}$ and solving the MaintainCluster problem on the resulting instances. The long edges however are reinserted into the contracted graph $\hat{H}$, and the actual lengths of the short edges are used in it as well. This approach unfortunately results in a rather large number of right-to-left edges with a large width in $\hat{H}$, as it may potentially include all long edges. In order to overcome this difficulty, we design an algorithm for the MaintainCluster problem that can handle arbitrary edge lengths, which adds an additional dimension of technical challenges.

The second main source of inefficiency in the algorithm of [CK24] is their use of Even-Shiloach trees in their algorithm for MaintainCluster, both in implementing the Matching Player in the Cut-Matching game, and in order to maintain short paths connecting all vertices of G to the vertices of the expander $\hat{G}$. It is immediate to see that the problem that the Matching Player needs to solve is essentially an instance of the RouteAndCut problem. We also observe that an algorithm for a variant of the RouteAndCut problem can be exploited in order to maintain the paths connecting all vertices of G to the vertices of $V(\hat{G})$. We abstract this as a new problem, that we call ConnectToCenters, and discuss it below. We believe that this problem and our algorithm for solving it are of independent interest. We remark that this reduction from the ConnectToCenters problem to the RouteAndCut problem requires that the algorithm for the RouteAndCut problem works for arbitrary congestion parameter $\eta \leq \Delta$, and this is the reason for our more general definition of the RouteAndCut problem.

To summarize, as already shown in previous work, in order to obtain an efficient algorithm for the RouteAndCut problem, it is enough to obtain an efficient algorithm for the MaintainCluster problem, and we observe that the opposite is also true: an efficient algorithm for the RouteAndCut problem implies an efficient algorithm for the MaintainCluster problem. This, however, creates a chicken-and-egg issue, where in order to solve one of the two problems efficiently, we need to design an efficient algorithm for the other. We overcome this barrier by using a recursive approach, that we describe next.

A recursive approach. We parameterize both the $\mathsf{RouteAndCut}$ and the $\mathsf{MaintainCluster}$ problem using a parameter $r > 0$. We say that an instance of the $\mathsf{MaintainCluster}$ problem on an n-vertex graph G with a distance parameter d is *r-restricted*, if $d \leq 2^{r\cdot\sqrt{\log n}}$. Consider now an instance of the $\mathsf{RouteAndCut}$ problem on an n-vertex graph H, with two subsets A, B of its vertices, and parameters Δ and η. It is not hard to see that, if $\mathcal{P}$ is any collection of $\Omega(\Delta/\operatorname{poly}\log n)$ paths connecting vertices of A to vertices of B, that cause vertex-congestion at most η, then a large fraction of the paths in $\mathcal{P}$ have length $\tilde{O}(n\eta/\Delta)$. We say that an instance of the $\mathsf{RouteAndCut}$ problem is *r-restricted* if $\frac{n\eta}{\Delta} \leq 2^{r\cdot\sqrt{\log n}}$. We show a straightforward algorithm for the 1-restricted $\mathsf{RouteAndCut}$ problem. Then for all $r \geq 1$, we show that an efficient algorithm for the r-restricted $\mathsf{RouteAndCut}$ problem implies an efficient algorithm for the r-restricted $\mathsf{MaintainCluster}$ problem. We also show that an efficient algorithm for the r-restricted $\mathsf{MaintainCluster}$ problem implies an efficient algorithm for the $(r+1)$-restricted $\mathsf{RouteAndCut}$ problem. Using induction on r, we then simultaneously obtain efficient algorithms for the $\mathsf{RouteAndCut}$ and the $\mathsf{MaintainCluster}$ problems for the entire range of values for the parameter r.

$\mathsf{ConnectToCenters}$ **Problem.** The $\mathsf{ConnectToCenters}$ problem is employed as a subroutine in the algorithm for the $\mathsf{MaintainCluster}$ problem, but we feel that it is interesting in its own right, as it seems to arise in many different settings. Suppose we are given a dynamic n-vertex graph G; for now we will assume that G undergoes an online sequence of edge deletions, but in fact our algorithm considers other updates, as described later. In addition to graph G, assume that we are given a parameter d, and a subset $T \subseteq V(G)$ of vertices that we call *centers*. The goal is to maintain, for every vertex $v \in V(G)$, a path $P(v)$ of length at most d, connecting v to some vertex of T. As the time progresses, some vertices may be deleted from T, but we are guaranteed that T always contains a large enough fraction of the vertices of G, e.g. $|T| \geq \Omega(|V(G)|/(d \operatorname{poly}\log n)$. In order to ensure that the deletion of edges from G does not impact too many paths in $\mathcal{P}^* = \{P(v) \mid v \in V(G)\}$, it is desirable that the paths cause a small edge-congestion (say at most $\tilde{O}(d)$), and for similar reasons it is desirable that every vertex $x \in T$ serves as an endpoint of relatively few such paths (say at most $\tilde{O}(d)$). At any time, the algorithm may compute a cut (A, B) of sparsity at most $O\left(\frac{\operatorname{poly}\log n}{d}\right)$, with $|A \cap T| \ll |A|$, after which the vertices of A are deleted from G and the algorithm continues. We note that whenever the by now standard paradigm of embedding an expander into the input graph G and then maintaining short paths connecting all vertices of G to the vertices of the expander is used, one essentially needs to solve a variant of the $\mathsf{ConnectToCenters}$ problem. So far this was typically done by employing Even-Shiloach trees, but this data structure becomes inefficient once the depth parameter d is sufficiently large. It is for this reason that we believe that our algorithm for the $\mathsf{ConnectToCenters}$ problem is of independent interest.

It is immediate to see that the initial collection $\mathcal{P}^* = \{P(v) \mid v \in V(G)\}$ of paths of length $O(d)$ each, connecting every vertex of G to some vertex of T, that cause edge-congestion $\tilde{O}(d)$, can be computed by employing an algorithm for the $\mathsf{RouteAndCut}$ problem. As edges are deleted from G, and vertices are deleted from T, some of the paths in $\mathcal{P}^*$ may be destroyed. Whenever a path $P(v) \in \mathcal{P}^*$ is destroyed (whether because its last endpoint is deleted from T, or because one of its edges is deleted from G), we say that vertex v becomes *disconnected*. We then need to *reconnect* all disconnected vertices to T. This, again, can be done by employing an algorithm for the $\mathsf{RouteAndCut}$ problem, but doing so directly may be very inefficient. Assume, for example, that we are given an algorithm $\mathcal{A}$ for the $\mathsf{RouteAndCut}$ problem, that, on an n-vertex graph G, has running time $O(n^{2+o(1)})$. Every time a subset of vertices of G becomes disconnected, we would need to employ this algorithm in order to reconnect them to T. However, it is possible that only a small number of vertices become disconnected at a time, and spending $\Theta(n^{2+o(1)})$ time to reconnect them each time is prohibitively expensive. A better approach seems to be to consider the set U of vertices that are currently connected, and the set U' of vertices that are currently disconnected. We could then attempt to route the vertices of U' to the vertices of U by constructing a new collection $\mathcal{Q} = \{Q(u) \mid u \in U'\}$ of paths, where each path $Q(u)$ connects u to some vertex of U; and then exploit the existing paths in $\{P(v) \mid v \in U\}$ in order to compute paths connecting every vertex of U' to the vertices of T. However, we cannot afford to spend $\Theta(n^2)$ time in order to compute the set $\mathcal{Q}$ of paths. On the other hand, intuitively, if $|U'| \ll U$, then we may not need to explore the entire graph G in order to compute the set $\mathcal{Q}$ of paths. In order to overcome this difficulty, we require that the algorithm for the RouteAndCut problem has running time roughly $n^{1+o(1)} \cdot (n - |B|)$, instead of $n^{2+o(1)}$. In particular, if the graph G is sufficiently dense and $|B|$ is sufficiently large, then this running time may be much smaller than $|E(G)|$. Our reduction from the $(r+1)$-restricted RouteAndCut problem to the r-restricted MaintainCluster problem follows the high-level approach of [BGS20] and [CK24]. However, this additional strict requirement on the efficiency of the algorithm for the RouteAndCut problem, in addition to the requirement that the algorithm should work for arbitrary values of the congestion parameter η, make the reduction more challenging and technical.

Assume now that we are given an algorithm $\mathcal{A}$ for the RouteAndCut problem, that, on an instance (G, A, B, Δ, η) has running time $n^{1+o(1)} \cdot (n - |B|)$, where $n = |V(G)|$. As described above, in order to implement our algorithm for the ConnectToCenters problem, whenever we are given a collection U' of vertices that are currently disconnected from T, we can now employ Algorithm $\mathcal{A}$ to construct a collection $\mathcal{Q}$ of paths connecting them to the vertices of U (the set of currently connected vertices), and then compose $\mathcal{Q}$ with the collection $\{P(v) \mid v \in U\}$ of paths to obtain the desired collection $\{P(v) \mid v \in U'\}$ of paths that reconnects the vertices of U' to T. Unfortunately, if we follow this approach, and keep appending paths to each other iteration after iteration, we may obtain paths whose lengths are prohibitively large. Instead, we follow a *layered* approach. For a parameter $\lambda = O(\log n)$, we maintain at all times a partition $(U_0, \ldots, U_\lambda)$ of the vertices of G into *layers*, where $U_0 = T$, and U_λ contains all vertices that are currently disconnected. For all $1 \leq i < \lambda$, we also maintain a collection $\mathcal{Q}_i = \{K(v) \mid v \in U_i\}$ of paths, where each path $K(v)$ connects a vertex $v \in U_i$ to a vertex in $\bigcup_{i'=0}^{i-1} U_{i'}$. The paths in each set $\mathcal{Q}_i$ have length at most $\frac{d}{4\lambda}$, and cause congestion $\tilde{O}(d)$ in G. By composing the paths from different sets $\mathcal{Q}_i$, we can obtain, for every vertex $v \in V(G)$, a path that connects it to some vertex of T. At a high level, for all $1 \leq i < \lambda$, we reconstruct layer U_i and the set $\mathcal{Q}_i$ of paths from scratch every time that roughly $2^{\lambda-i}$ new vertices become disconnected, and we also ensure that $|U_i| \leq 2^{\lambda-i}$ holds. Therefore, when we employ the algorithm for the RouteAndCut problem in order to reconnect the vertices of U_i, in the resulting instance of RouteAndCut, $n - |B| \leq 2^{\lambda-i}$ holds, and the running time of the algorithm is roughly $n^{1+o(1)} \cdot 2^{\lambda-i}$. Therefore, as index i becomes smaller, the running time of the RouteAndCut algorithm that is used to reconnect the vertices of U_i increases. However, for smaller values of index i, we need to reconstruct the set U_i of vertices and the set $\mathcal{Q}_i$ of paths less often. This eventually leads to the desired $n^{2+o(1)}$ running time.

Edge-deletion versus edge-length-increase updates. We would like to highlight here what we feel is a somewhat surprising insight from our work, that may appear minor at first sight, but we believe that it may be useful in other problems. Consider the following two settings for dynamic graphs: the first one is the standard decremental setting, where edges are deleted from the input graph G as the time progresses. The second setting is a somewhat more unusual one, where the only updates that are allowed in the input graph G is the doubling of the lengths of its edges; we refer to this type of updates as edge-length-increases. Generally, it is not hard to see that both models are roughly equivalent. Indeed, in order to simulate edge-length-increases in the standard decremental setting, we can simply create a large number of copies of each edge e of various lengths, and then, as the length of e increases, some of these copies are deleted. The reduction in the other direction is also immediate: we can simulate the deletion of an edge e from G by repeatedly doubling its length, until it becomes very high.

The dynamic graphs that arise from using the MWU framework typically only undergo edge-length-increase updates, which are then typically implemented as edge-deletions, in order to reduce the problem to the more standard decremental SSSP, as described above. However, the edge-length-increases that the input graph G undergoes under this implementation of the MWU method are rather well-behaved: specifically, the lengths of the edges are only increased moderately, and all length increases occur on the edges that participate in the paths that the algorithm returns in response to queries. We observe that the resulting SSSP problem appears to be easier if we work with edge-length-increase updates directly, instead of the more traditional approach of transforming them into edge-deletion updates.

In order to illustrate this, consider the following simple scenario: we are given a graph G, and initially the length of every edge in G is 1. Assume also that we have computed another graph X (it may be convenient to think of X as an expander), and embedded X into G via paths of length at most d, that cause edge-congestion at most η. If some edge e is deleted from G, then every edge $\hat{e} \in E(X)$, whose embedding path $Q(\hat{e})$ uses e, must be deleted from X as well. Therefore, the deletion of a single edge from G may lead to the deletion of η edges from X. Assume now that, instead, the edges of G only undergo increases in their lengths, where the length of each edge may be iteratively doubled, but the total increase in the lengths of all edges is moderate. If the length of an edge $e \in E(G)$ is doubled, then for every edge $\hat{e} \in E(X)$ whose embedding path $Q(\hat{e})$ uses e, the length of $Q(\hat{e})$ increases only slightly, and so there is no need to delete $\hat{e}$ from $E(X)$. We can usually wait until the length of the path $Q(\hat{e})$ increases significantly before edge $\hat{e}$ needs to be deleted from X. As mentioned already, in instances arising from the MWU framework, the total increase in the lengths of all edges in G over the course of the entire algorithm is usually not very large, and in particular most edges whose lengths are doubled are short. This allows us to maintain the expander X and its embedding into G over the course of a much longer sequence of updates to G. This is one of the insights that allowed us to obtain a more efficient algorithm for the MaintainCluster problem.

To summarize, our algorithm departs from the algorithm of [CK24] in the following key aspects. First, we use the MWU method to reduce the RouteAndCut problem to s-t-SP in dynamic graphs that undergo edge-length-increases instead of edge-deletion updates. Second, our algorithm for the RouteAndCut problem has running time that significantly decreases when the set B of vertices contains almost all vertices of G; in some cases the running time may even be lower than $|E(G)|$. We extend the MaintainCluster problem to handle arbitrary edge lengths, but unlike [CK24] we only allow edge-length-increases, instead of edge-deletion updates. We design an algorithm for the ConnectToCenters problem, that we believe is of independent interest, and that can be viewed as reducing the MaintainCluster problem to RouteAndCut. Lastly, we use a recursive approach, in which algorithms for RouteAndCut rely on algorithms for MaintainCluster and vice versa, by parametrizing both problems with an auxiliary parameter r, and then inductively develop algorithms for both problems for the entire range of r.

1.2 Organization

We start with preliminaries in Section 2. We then provide a high-level overview of our algorithm in Section 3, where we also define the RouteAndCut and the s-t-SP problems, state our main results for them, and describe the modification of the MWU method that we use. In Section 4 we define the MaintainCluster problem and state our main result for it. We also describe our recursive approach, state the theorems describing the reductions between the MaintainCluster and the RouteAndCut problems, and complete the algorithm for the Maximum Bipartite Matching problem using them. Then in Section 5 we describe our reduction from the $(r+1)$-restricted RouteAndCut problem to the r-

restricted MaintainCluster problem. In Section 6 we describe our algorithm for the ConnectToCenters problem, and in Section 7 we complete our reduction from the r-restricted MaintainCluster problem to the r-restricted RouteAndCut problem.

2 Preliminaries

In this paper we work with both directed and undirected graphs. By default graphs are not allowed to contain parallel edges or self-loops. Graphs with parallel edges are explicitly referred to as *multigraphs.*

Congestion of Paths and Flows. Let G be a graph with capacities $c(e) \geq 0$ on edges $e \in E(G)$, and let $\mathcal{P}$ be a collection of simple paths in G. We say that the paths in $\mathcal{P}$ cause *edge-congestion* η if every edge $e \in E(G)$ participates in at most $\eta \cdot c(e)$ paths in $\mathcal{P}$. When edge capacities are not explicitly given, we assume that they are unit. If every edge of G belongs to at most one path in $\mathcal{P}$, then we say that the paths in $\mathcal{P}$ are *edge-disjoint.*

Similarly, if we are given a flow value $f(e) \geq 0$ for every edge $e \in E(G)$, we say that flow f causes *edge-congestion* η if, for every edge $e \in E(G)$, $f(e) \leq \eta \cdot c(e)$ holds. If $f(e) \leq c(e)$ holds for every edge $e \in E(G)$, we may say that f causes *no edge-congestion.*

Assume now that we are given a graph G, a pair s, t of its vertices, and a collection $\mathcal{P}$ of simple paths connecting s to t. We say that the paths in $\mathcal{P}$ cause *vertex-congestion* η if every vertex $v \in V(G) \setminus \{s, t\}$ participates in at most η paths. If the paths in $\mathcal{P}$ cause vertex-congestion 1, then we say that they are *internally vertex-disjoint.*

Throughout the paper, by default, when we talk about congestion caused by a set of paths or by a flow, we mean **edge-congestion**.

Embeddings of Graphs. Let H, G be two graphs with $V(H) \subseteq V(G)$. An *embedding* of H into G is a collection $\mathcal{P} = \{P(e) \mid e \in E(H)\}$ of simple paths in G, where for every edge $e = (u, v) \in E(H)$, path $P(e)$ connects u to v in G. 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'; equivalently, it is the edge-congestion that the set $\mathcal{P}$ of paths causes in G.

Cuts, Sparsity and Expanders for Directed Graphs. Let G be a directed graph. A *cut* in G is a an ordered pair (A, B) of subsets of vertices of G, such that $A \cap B = \emptyset$, $A \cup B = V(G)$, and $A, B \neq \emptyset$. Note that we do not require that $|A| \leq |B|$ in this definition. The *sparsity* of the cut is:

$$\Phi(A, B) = \frac{|E(A, B)|}{\min\{|A|, |B|\}}.$$

We say that a directed graph G is a φ-expander, for a given value $0 < \varphi < 1$, if the sparsity of every cut in G is at least φ. Notice that, if G is a φ-expander, then for every partition (A, B) of its vertices, both $|E(A, B)| \geq \varphi \cdot \min\{|A|, |B|\}$ and $|E(B, A)| \geq \varphi \cdot \min\{|A|, |B|\}$ hold.

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 for undirected graphs was shown in [CGL$^+$20].

Theorem 2.1 (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 $n > 1$, constructs a directed graph H_n*

with $|V(H_n)| = n$, such that H_n is an α_0-expander, and every vertex in H_n has at most 9 incoming and at most 9 outgoing edges. The running time of the algorithm is $O(n)$.

The above theorem was proved in [CGL+20] for undirected expanders, with maximum vertex degree in the resulting expander bounded by 9. It is easy to adapt this result to directed expanders: if a graph G is an undirected φ-expander, and a directed graph G' is obtained from G by replacing every edge with a pair of bi-directed edges, then it is easy to verify that G' is a directed φ-expander.

We will also use the following simple observation, that shows that a slight modification of an expander graph still results in a (somewhat weaker) expander. Similar observations were shown in previous work for undirected graphs (see e.g. Observation 2.3 in [CGL+20]).

Observation 2.2 *Let H be a directed n-vertex graph, that is a φ-expander, for some parameter $0 < \varphi < 1$. Let H' be a graph that is obtained from H as follows. We select an arbitrary set X of $n' \leq n$ vertices of H, add a set Y of n' new vertices to H, and add two matchings of cardinality n' to H: a matching $M' \subseteq X \times Y$ and a matching $M'' \subseteq Y \times X$. Then the resulting graph H', defined over $(n + n')$ vertices, is a $\varphi/3$-expander.*

Proof: Consider any cut (A, B) in H'. Assume first that $|A \cap Y| \geq \frac{2|A|}{3}$. Recall that every vertex $y \in Y$ has one edge $e_y = (y, y') \in M''$ with $y' \in X$. Let $Y_1 \subseteq A \cap Y$ denote the set of all vertices $y \in A \cap Y$ whose corresponding vertex y' lies in A, and let Y_2 denote the set of all remaining vertices of $A \cap Y$. Since $|A \setminus Y| \leq \frac{|A \cap Y|}{2}$, it must be the case that $|Y_2| \geq |Y_1| \geq \frac{|A|}{3}$. But then the edges of M'' that are incident to the vertices of Y_2 belong to $E_{H'}(A, B)$, and so $|E_{H'}(A, B)| \geq |Y_2| \geq \frac{|A|}{3}$.

Similarly, if $|B \cap Y| \geq \frac{2|B|}{3}$, then, by using a similar argument with the edges of the matching M' instead of M'', we get that $|E_{H'}(A, B)| \geq \frac{|B|}{3}$. Therefore, if either $|A \cap Y| \geq \frac{2|A|}{3}$, or $|B \cap Y| \geq \frac{2|B|}{3}$ hold, then the sparsity of the cut (A, B) is at least $\frac{1}{3}$.

It now remains to consider the case where both $|A \cap Y| < \frac{2|A|}{3}$ and $|B \cap Y| < \frac{2|B|}{3}$ hold. Let $A' = A \setminus Y$ and $B' = B \setminus Y$. Notice that $|A'| \geq |A|/3$ and $|B'| \geq |B|/3$ must hold. Then (A', B') is a cut in graph H, and, since H is a φ-expander, $|E_H(A', B')| \geq \varphi \cdot \min\{|A'|, |B'|\} \geq \frac{\varphi}{3} \cdot \min\{|A|, |B|\}$. Since $E_H(A', B') \subseteq E_{H'}(A, B)$, we get that the sparsity of the cut (A, B) in H' is at least $\frac{\varphi}{3}$. □

The following simple observation follows from the standard ball-growing technique, and similar observations were used in many previous works, including in [CK24], where a proof is provided for completeness.

Observation 2.3 (Observation 2.2 in full version of [CK24]) *There is a deterministic algorithm, that, given a directed n-vertex graph $G = (V, E)$ with unit edge lengths and maximum vertex degree at most $\delta_{\max}$, together with two vertices $x, y \in V$ with $\mathsf{dist}_G(x, y) \geq d$ for some parameter $d \geq 64\delta_{\max} \log n$, computes a cut (A, B) of sparsity at most $\varphi = \frac{32\delta_{\max} \log n}{d}$ in G. The running time of the algorithm is $O(\delta_{\max} \cdot \min\{|A|, |B|\})$.*

We will also use the following simple observation a number of times. Intuitively, the observation allows us to convert a sequence of small sparse cuts into a single balanced sparse cut; a proof of this observation appears in [CK24].

Observation 2.4 (Observation 2.3 in full version of [CK24]) *There is a deterministic algorithm, whose input consists of a directed n-vertex graph G, parameters $0 < \alpha < 1$ and $0 \leq \varphi \leq 1$, and a sequence $\mathcal{X} = (X_1, X_2, \ldots, X_k)$ of subsets of vertices of G, such that the following hold:*

- *Every vertex of $V(G)$ lies in exactly one set in $\{X_1, \ldots, X_k\}$;*

- $\max_i \{|X_i|\} = \alpha n$; *and*
- *for all* $1 \leq i < k$, *if we denote by* $X'_i = \bigcup_{j=i+1}^{k} X_j$, *then either* $|E(X_i, X'_i)| \leq \varphi \cdot |X_i|$, *or* $|E(X'_i, X_i)| \leq \varphi \cdot |X_i|$.

The algorithm computes a cut (A, B) *in* G *with* $|A|, |B| \geq \min\left\{\frac{1-\alpha}{2} \cdot n, \frac{n}{4}\right\}$, *and* $|E_G(A, B)| \leq \varphi \cdot \sum_{i=1}^{k-1} |X_i|$. *The running time of the algorithm is* $O(|E(G)|)$.

Chernoff Bound. We use the following standard version of Chernoff Bound (see. e.g., [DP09]).

Lemma 2.5 (Chernoff Bound) *Let* $X_1, \ldots, X_n$ *be independent randon variables taking values in* $\{0, 1\}$. *Let* $X = \sum_{1 \leq i \leq n} X_i$, *and let* $\mu = \mathbf{E}[X]$. *Then for any* $t > 2e\mu$,

$$\Pr\Big[X > t\Big] \leq 2^{-t}.$$

Additionally, for any $0 \leq \delta \leq 1$,

$$\Pr\Big[X < (1 - \delta) \cdot \mu\Big] \leq e^{-\frac{\delta^2 \cdot \mu}{2}}.$$

2.1 Even-Shiloach Tree for Weighted Directed Graphs

We use the generalization of the Even-Shiloach data structure of [ES81, Din06] to weighted directed graphs, due to [Kin99], who extended similar results of [HK95] for unweighted directed graphs.

Theorem 2.6 ([Kin99], see Section 2.1) *There is a deterministic algorithm, whose input consists of a directed graph* H *with integral edge lengths, a source vertex* $s \in V(H)$, *and a distance parameter* $d \geq 1$. *The algorithm maintains a shortest-path tree* $\tau \subseteq H$ *rooted in* s, *up to depth* d. *In other words, for every vertex* $v \in V(H)$ *with* $\mathsf{dist}_H(s, v) \leq d$, *vertex* v *lies in the tree* τ, *and the length of the shortest* s*-*v *path in* τ *is equal to that in* H. *The data structure also maintains, for all* $v \in V(\tau)$, *the current value* $\mathsf{dist}_H(s, v)$. *The total update time of the algorithm is* $O(|E(H)| \cdot d)$.

We will sometimes refer to the data structure from Theorem 2.6 as $\mathsf{ES\text{-}Tree}(H, s, d)$.

2.2 The Adjacency-List Representation of Graphs

In some of the problems that we consider, the algorithm may not need to explore the entire input graph G, and in some cases, it is important that the running time of the algorithm is strictly smaller than $|E(G)|$. As an example, consider the following routing problem: we are given as input a directed graph G, and two disjoint sets A,B of its vertices. The goal is to compute a largest-cardinality collection $\mathcal{P}$ of paths, where each path $P \in \mathcal{P}$ connects a vertex of A to a vertex of B, the paths in $\mathcal{P}$ cause low edge-congestion, and every vertex of $A \cup B$ serves as an endpoint of at most one path in $\mathcal{P}$. We emphasize that vertices of $A \cup B$ may serve as inner vertices on paths in $\mathcal{P}$, and it is possible that some vertex $v \in A \cup B$ serves both as an endpoint of some path in $\mathcal{P}$, and as an inner vertex on several other paths. Consider now a scenario where almost all vertices of G lie in B. In such a case, one can show that there is an optimal solution $\mathcal{P}$ to the problem, such that only a small number of vertices of B participate in the paths in $\mathcal{P}$. In other words, in order to compute such a collection of paths, it is not necessary to explore the entire graph G. In such cases, we will often be ineterested in algorithms with running time roughly $O(n^{1+o(1)} \cdot (n - |B|))$, that may be significantly smaller than $|E(G)|$, if graph G is sufficiently dense. Therefore, we will need to provide the input graph G to the algorithm in a way that allows it to achieve this low running time, and does not require spending $\Omega(|E(G)|)$ time in order to read the entire input.

In such cases, we will often require that the (directed) input graph G is provided to the algorithm in the adjacency-list model: we are given two arrays OUT and IN of length n each. For all $1 \leq i \leq n$, entry OUT$[i]$ contains a pointer to a linked-list, that contains all edges in set $\delta^+(v_i)$, that is, all edges that leave the ith vertex of G. Similarly, IN$[i]$ contains a pointer to a linked-list of all edges in $\delta^-(v_i)$ – all edges that enter the ith vertex of G. If the edges of G have lengths or weights, then these lengths or weights also appear in the corresponding lists $\delta^-(v_i)$ and $\delta^+(v_i)$, where these edges are listed.

Typically, if Alg is an algorithm for the routing problem mentioned above, then it will only access the entries OUT$[i]$, IN$[i]$ of a small subset of the vertices $v_i \in V(G)$, allowing us to achieve a running time that is significantly lower than $\Theta(|E(G)|)$. Specifically, it is sufficient that the algorithm reads the entries corresponding to the vertices in $V(G) \setminus B$, and of a small subset of vertices in B. This will allow us to obtain running times for such routing problems roughly of the form $O(|V(G)|^{1+o(1)} \cdot (|V(G)| - |B|))$, which may be significantly lower than $\Theta(|E(G)|)$ if set B contains almost all vertices of G, and G is sufficiently dense.

We note that providing the graph G as input to such an algorithm Alg may require time $O(|E(G)| + |V(G)|)$, in order to construct the adjacency-list representation of the input graph G. However, in a typical application, we will construct the adjacency-list representation of the input graph G once, and then apply Algorithm Alg for solving the routing problem on G multiple times, with different vertex sets A and B, possibly with small alterations to the graph G between such executions of Alg. Therefore, we will not need to construct the adjacency-list representation of the input graph G from scratch every time Algorithm Alg is called.

In some cases, given as input a directed graph G in the adjacency-list representation, our routing algorithm will need to work with the graph $\overleftarrow{G}$, which is obtained from G by reversing the directions of all its edges. In such cases, we can modify the adjacency-list representation of G to obtain an adjacency-list representation of $\overleftarrow{G}$, by simply switching the arrays OUT and IN.

Finally, given a directed graph G and a vertex v in G, we will denote by $\delta^+_G(v)$, the set of edges leaving v, and by $\delta^-_G(v)$, the set of edges entering v.

2.3 Overview of the $\mathcal{ATO}$ framework

We now provide an overview of the Approximate Topological Order ($\mathcal{ATO}$) framework that was developed in [BGWN20] and later used in [BGS20]. A very slight modification of this framework was also used in [CK24]. We follow here the presentation of [CK24], though we further adapt it to allow for presence of a special cluster, that contains most of the vertices of the underlying graph.

Let $G = (V, E)$ be a directed n-vertex graph with two special vertices s and t and non-negative lengths $\ell(e)$ on edges $e \in E$, that undergoes edge-length-increase updates during a time interval $\mathcal{T}$, that we refer to as the *time horizon*. Throughout, we refer to a collection $I \subseteq \{1, \ldots, n\}$ of consecutive integers as an *interval*. The *size* of the interval I, denoted by $|I|$, is the number of distinct integers in it. An $\mathcal{ATO}$ consists of the following components:

- a partition $\mathcal{X}$ of the set V of vertices of G into non-empty subsets, such that there is a set $S = \{s\}$ and $T = \{t\}$ in $\mathcal{X}$;
- (optional) a single set $U \in \mathcal{X}$ that we refer to as a *special cluster*, together with an integer $1 \leq \gamma \leq |U|$;

If special cluster U is defined, then all other sets of $\mathcal{X}$ are called *regular clusters*. If the special cluster is undefined, then all sets of $\mathcal{X}$ are called regular clusters, and we set $\gamma = 1$; and

- a map ρ, that maps every set $X \in \mathcal{X}$ to an interval $I_X \subseteq \{1, \ldots, n - \gamma + 1\}$. The size of the interval I_X must be $|X|$ if X is a regular cluster, and $|X| - \gamma + 1$, if it is a special cluster. We require that all intervals in $\{I_X \mid X \in \mathcal{X}\}$ are mutually disjoint.

For convenience, we will denote an $\mathcal{ATO}$ by $(\mathcal{X}, \rho, U, \gamma)$, where the special cluster U may be undefined. Both the partition $\mathcal{X}$ and the map ρ evolve over time. We denote by $\mathcal{X}^{(\tau)}$ the partition $\mathcal{X}$ at time τ, and by $\rho^{(\tau)}$ the map ρ at time τ. For every set $X \in \mathcal{X}^{(\tau)}$, we denote by $I_X^{(\tau)}$ the interval I_X associated with X at time τ. If the special cluster U is defined in the initial $\mathcal{ATO}$, then we require that $U \in \mathcal{X}$ holds at all times, though some vertices may be deleted from U as the time progresses, and, additionally, $|U| \geq \gamma$ must hold at all times.

The only type of changes that are allowed to the $\mathcal{ATO}$ over time is the **splitting** of the clusters $X \in \mathcal{X}$. In order to split a set $X \in \mathcal{X}$, we select a subset $Y \subseteq X$ of vertices (if X is a special cluster, then $|X \setminus Y| \geq \gamma$ must hold, while otherwise we require that $X \setminus Y \neq \emptyset$). We add Y to $\mathcal{X}$ as a new cluster, and we delete the vertices of Y from X. We also partition interval I_X into two disjoint subintervals: one of these subintervals, of length $|Y|$, becomes the interval I_Y that is associated with the new cluster Y. The other subinterval, whose length is $|X|$ if X is a regular cluster, and $|X| - \gamma + 1$ if it is a special cluster, becomes the new interval I_X associated with the cluster X. Note that the total length of the two new intervals is equal to the length of the original interval I_X.

Notice that an $\mathcal{ATO}$ $(\mathcal{X}, \rho, U, \gamma)$ naturally defines a left-to-right ordering of the intervals in $\{I_X \mid X \in \mathcal{X}\}$: we will say that interval I_X lies to the left of interval $I_{X'}$, and denote $I_X \prec I_{X'}$ iff every integer in interval I_X is smaller than every integer in $I_{X'}$. We will also sometimes denote $X \prec X'$ in such a case. Notice also that, if we are given an ordering $\sigma = (X_1, X_2, \ldots, X_k)$ of the sets in $\mathcal{X}$, then this ordering naturally defines the map ρ, where for all $1 \leq i \leq k$, the first integer in I_{X_i} is $\sum_{j=1}^{i-1} |X_j| + 1$ if none of the clusters in $\{X_1, \ldots, X_{i-1}\}$ is special, and it is $\sum_{j=1}^{i-1} |X_j| - \gamma + 2$ otherwise. For convenience, we will denote the map ρ associated with the ordering σ by $\rho(\sigma)$.

Given a vertex $v \in V$, we will sometimes denote by X^v the set of $\mathcal{X}$ that contains v. Consider now some edge $e = (u, v)$ in graph G, and assume that $X^u \neq X^v$. We say that e is a *left-to-right* edge if $X^u \prec X^v$, and we say that it is a *right-to-left* edge otherwise.

Lastly, we denote by $H = G_{|\mathcal{X}}$ the *contracted graph* associated with G and $\mathcal{X}$ – the graph that is obtained from G by contracting every set $X \in \mathcal{X}$ into a vertex v_X, that we may call a *supernode*, to distinguish it from other vertices of G. We keep parallel edges but discard any self-loops.

Intuition. For intuition, we could let $\mathcal{X}$ be the set of all strongly connected components of the graph G. The corresponding contracted graph $H = G_{|\mathcal{X}}$ is then a DAG, so we can define a topological order of its vertices. This order, in turn, defines an ordering σ of the sets in $\mathcal{X}$, and we can then let the corresponding $\mathcal{ATO}$ be $(\mathcal{X}, \rho(\sigma), U, \gamma)$, where U is undefined and $\gamma = 1$. Notice that in this setting, all edges of G that connect different sets in $\mathcal{X}$ are left-to-right edges. However, we will not follow this scheme. Instead, we will ensure that every set $X \in \mathcal{X}$ corresponds to some expander-like graph, and we will allow both left-to-right and right-to-left edges with respect to the $\mathcal{ATO}$. It would still be useful to ensure that the corresponding contracted graph H is close to being a DAG, since there are good algorithms for SSSP on DAG's and DAG-like graphs. Therefore, we will try to limit both the number of right-to-left edges, and their "span", that we define below.

Skip and Span of Edges. For every edge $e = (u, v) \in E(G)$ we define two values: the *skip* of edge e, denoted by $\mathrm{skip}(e)$, and the *span* of e, denoted by $\mathrm{span}(e)$. Intuitively, the skip value of an edge (u, v) will represent the "distance" between the intervals I_{X^u} and I_{X^v}. Non-zero span values are only assigned to right-to-left edges. The span value also represents the distance between the corresponding intervals, but this distance is defined differently. We now formally define the span and skip values for each edge, and provide an intuition for them below.

Consider some edge $e = (x, y) \in E(G)$. Let $X, Y \in \mathcal{X}$ be the sets containing x and y, respectively. If $X = Y$, then $\mathrm{skip}(e) = \mathrm{span}(e) = 0$.

Assume now that $X \prec Y$, that is, e is a left-to-right edge. In this case, we set $\mathrm{span}(e) = 0$, and we let $\mathrm{skip}(e)$ be the distance between the last endpoint of I_X and the first endpoint of Y. In other words, if the largest integer in I_X is Z_X, and the smallest integer in Y is A_Y, then $\mathrm{skip}(e) = A_Y - Z_X$. Note that, as the algorithm progresses, sets X and Y may be partitioned into smaller subsets. It is then possible that $\mathrm{skip}(e)$ may grow over time, but it may never decrease.

Lastly, assume that $Y \prec X$, so e is a right-to-left edge. In this case, we set $\mathrm{skip}(e)$ to be the distance between the right endpoint of I_Y and the left endpoint of I_X, and we let $\mathrm{span}(e)$ be the distance between the left endpoint of I_Y and the right endpoint of I_X. In other words, if A_X and Z_X are the smallest and the largest integers in X respectively, and similarly A_Y and Z_Y are the smallest and the largest integers in Y respectively, then $\mathrm{skip}(e) = A_X - Z_Y$ and $\mathrm{span}(e) = Z_X - A_Y$ (see Figure 1). Notice that, as the algorithm progresses, and sets X and Y are partitioned into smaller subsets, the value $\mathrm{span}(e)$ may only decrease, while the value $\mathrm{skip}(e)$ may only grow. Moreover, $\mathrm{skip}(e) \leq \mathrm{span}(e)$ always holds.

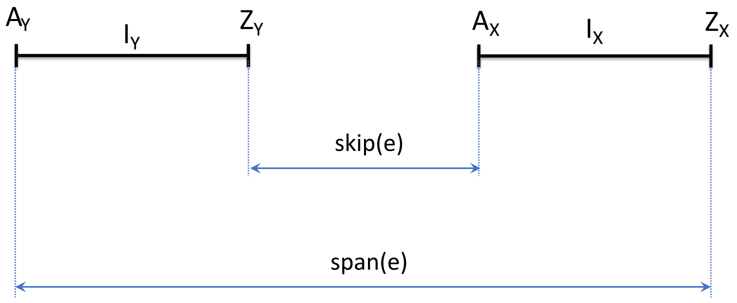

Figure 1: Skip and span values of a right-to-left edge.

For intuition, consider again the setting where $\mathcal{X}$ represents the strongly connected components of G, and $\rho = \rho(\sigma)$, where σ is an ordering of the sets in $\mathcal{X}$ corresponding to a topological ordering of the vertices of $H = G_{|\mathcal{X}}$. Then for any path P in G, $\sum_{e \in E(P)} \mathrm{skip}(e) \leq n$ must hold. This is because the path must traverse the sets of $\mathcal{X}$ in their left-to-right order. Consider now inserting a single right-to-left edge $e = (x, y)$ into G, so $x \in X$, $y \in Y$, and $Y \prec X$. In this case, path P will generally traverse the sets in $\mathcal{X}$ from left to right, but it may use the edge e in order to "skip" back. It is not hard to see that $\sum_{e' \in E(P)} \mathrm{skip}(e') \leq n + \mathrm{skip}(e) + \mathrm{span}(e)$ now holds, since, by traversing the edge e, path P is 'set back' by at most $\mathrm{span}(e)$ units. In other words, path P traverses the sets in $\mathcal{X}$ in their left-to-right order, except when it uses the edge e to "skip" to the left. The length of the interval it "skips" over is bounded by $\mathrm{span}(e)$, so this skip adds at most $\mathrm{skip}(e) + \mathrm{span}(e) \leq 2\,\mathrm{span}(e)$ to $\sum_{e' \in E(P)} \mathrm{skip}(e')$.

Using the same intuitive reasoning, if we allow graph G to contain an arbitrary number of right-to-left edges, then for any simple path P, $\sum_{e \in E(P)} \mathrm{skip}(e) \leq n + \sum_{e \in E(G)} 2\,\mathrm{span}(e)$ holds. However, if we are given any collection $\mathcal{P}$ of edge-disjoint paths in G, then there must be some path $P \in \mathcal{P}$ with $\sum_{e\in E(P)} \text{skip}(e) \leq n + \frac{\sum_{e\in E(G)} 2\,\text{span}(e)}{|\mathcal{P}|}$, since every edge $e \in E(G)$ may contribute the value $2\,\text{span}(e)$ to at most one path in $\mathcal{P}$. We will use this fact extensively.

From $\mathcal{ATO}$ to SSSP in DAG-like Graphs. Like the algorithm of [BGS20] for decremental SSSP, our algorithm relies on two main technical tools. The first tool allows us to maintain the partition $\mathcal{X}$ of $V(G)$ over time. Intuitively, we will ensure that, for every set $X \in \mathcal{X}$, the corresponding induced subgraph $G[X]$ of G is in some sense expander-like. In particular, we will be able to support short-path queries between vertices of X efficiently. When $G[X]$ is no longer an expander-like graph, we will compute a sparse cut (A, B) in this graph, and we will replace X with A and B in $\mathcal{X}$ (though formally we will think about this process as splitting the cluster X: we add a new cluster, A or B, to $\mathcal{X}$, and then delete the vertices of this new cluster from X). Some of the edges connecting the sets A and B may become right-to-left edges with respect to the current $\mathcal{ATO}$ $(\mathcal{X}, \rho)$ that we maintain. We will control the number of such edges and their span values by ensuring that the cut is φ-sparse, for an appropriately chosen parameter φ.

Given a dynamically evolving $\mathcal{ATO}$ $(\mathcal{X}, \rho, U, \gamma)$ for graph G, we can now construct the corresponding contracted graph $H = G_{|\mathcal{X}}$. For convenience, the supernodes of H representing the sets $S = \{s\}$ and $T = \{t\}$ of $\mathcal{X}$ are denoted by s and t, respectively. Notice that, as the time progresses, graph H undergoes, in addition to edge-length-increase updates (that we will implement as edge-deletions using standard methods) another type of updates, that we call *vertex-splitting*: whenever some set $X \in \mathcal{X}$ undergoes a splitting, and a new cluster $Y \subseteq X$ is inserted into $\mathcal{X}$, we need to insert a supernode corresponding to Y into H, and then possibly delete some edges that are incident to v_X from H. We will say that the new supernode v_Y was *split off* from the supernode v_X. Consider now any edge $e = (x, y) \in E(G)$, and assume that $x \in Z$ and $y \in Z'$ for some $Z, Z' \in \mathcal{X}$ with $Z \neq Z'$. The length of the edge e in graph H remains the same: $\ell(e)$. We will also associate, with every edge $e \in E(H)$, a *weight* $w(e)$, that is set to be the smallest integral power of 2 that is at least skip(e). As observed already, as the algorithm progresses, the value skip(e) may grow. In order to avoid inserting edges into H overtime, we will create a number of parallel edges with different $w(e)$ values corresponding to every edge of G whose endpoints lie in different clusters of $\mathcal{X}$. Specifically, let $e = (x, y)$ be any edge, with $x \in X$, $y \in Y$, and $X \neq Y$. For every integer $0 \leq i \leq \lceil \log n \rceil$, such that $2^i \geq \text{skip}(e)$, we add an edge $e_i = (v_X, v_Y)$ to H, with length $\ell(e)$ and $w(e_i) = 2^i$. It is easy to verify that this construction ensures that, for every vertex $v \in V(H) \setminus \{t\}$ and integer $0 \leq i \leq \lceil \log n \rceil$, the number of vertices in the set $\left\{u \in V(H) \mid \exists e = (v, u) \in E(H) \text{ and } w(e) = 2^i\right\}$ is bounded by 2^{i+2}. Additionally, by carefully controlling the total span of all right-to-left edges in graph G, we will ensure that there is always a short s-t path P in H, such that $\sum_{e\in E(P)} w(e)$ is relatively small. These two properties are crucial in obtaining a fast algorithm for computing short s-t paths in H, as it undergoes edge-deletion and supernode-splitting updates.

The $\mathcal{ATO}$ framework, combined with an algorithm for maintaining expander-like graphs, gives rise to the problem of repeately computing short s-t paths in the contracted graph H, as it undergoes the updates that we have outlined above. In [CK24] a problem call DAG-like SSSP was defined, that abstracts the above problem. They also provided an algorithm for it, which is essentially identical to the algorithm of [BGWN20]; a similar algorithm was used in [BGS20]. We now provide the definition of the DAG-like SSSP problem from [CK24], and summarize their algorithm for it.

2.4 SSSP in DAG-like Graphs

The presentation in this section is almost identical to the definition of the DAG-like SSSP problem from [CK24]. The input to the DAG-like SSSP problem is a directed multigraph $G = (V, E)$, with two designated vertices s and t, a distance parameter $d > 0$, a precision parameter $0 < \epsilon \leq 1/8$, such that $1/\epsilon$ is an integer, and another parameter $\Gamma > 0$. Graph G may have parallel edges, but no self-loops. Every edge $e \in E$ of G is associated with an integral *length* $\ell(e) > 0$, and an additional *weight* $w(e)$, that must be an integral power of 2 between 1 and $2^{\lceil \log n \rceil}$. The graph undergoes the following two types of updates:

- **Edge deletion:** given an edge $e \in E$, delete e from G; and
- **Vertex splitting:** in this operation, we are given as input a vertex $v \notin \{s, t\}$ in the current graph, and new vertices $u_1, \ldots, u_k$ that need to be inserted into G; we may sometimes say that vertices $u_1, \ldots, u_k$ are *split-off from* v. In addition to vertex v and vertices $u_1, \ldots, u_k$, we are given a new collection E' of edges that must be inserted into G, and for every edge $e \in E'$, we are given an integral length $\ell(e) > 0$ and a weight $w(e)$ that must be an integral power of 2 between 1 and $2^{\lceil \log n \rceil}$. For each such edge $e \in E'$, either both endpoints of e lie in $\{u_1, \ldots, u_k, v\}$; or one endpoint lies in $\{u_1, \ldots, u_k\}$, and the other in $V(G) \setminus \{v, u_1, \ldots, u_k\}$. In the latter case, if $e = (u_i, x)$, then edge $e' = (v, x)$ must currently lie in G, and both $w(e') = w(e)$ and $\ell(e') = \ell(e)$ must hold. Similarly, if $e = (x, u_i)$, then edge $e' = (x, v)$ must currently lie in G, and both $w(e') = w(e)$ and $\ell(e') = \ell(e)$ must hold.

We let $\mathcal{T}$ be the *time horizon* of the algorithm – that is, the time interval during which the graph G undergoes all updates. We denote by n and by m the total number of vertices and edges, respectively, that were ever present in G during $\mathcal{T}$. We assume w.l.o.g. that for every vertex v that was ever present in G, some edge was incident to v at some time during the update sequence, so $n \leq O(m)$.

For every vertex $v \in V(G) \setminus \{t\}$ and integer $0 \leq i \leq \lceil \log n \rceil$, we define a set:

$$U_i(v) = \left\{ u \in V(G) \mid \exists e = (v, u) \in E(G) \text{ with } w(e) = 2^i \right\}.$$

Note that as the algorithm progresses, each such set $U_i(v)$ may change, with vertices both leaving and joining the set. We require that the following property must hold at all times $\tau \in \mathcal{T}$:

S1. For every vertex $v \in V(G) \setminus \{t\}$, for every integer $0 \leq i \leq \lceil \log n \rceil$, $|U_i(v)| \leq 2^{i+2}$.

As graph G undergoes edge-deletion and vertex-splitting updates, at any time the algorithm may be asked a query path-query. It is then required to return a **simple** path P connecting s to t, whose length $\sum_{e \in E(P)} \ell(e)$ is bounded by $(1 + 10\epsilon)d$, in time $O(|E(P)|)$.

Lastly, the algorithm may return FAIL at any time and halt. It may only do so if the current graph G does not contain any s-t path P with $\sum_{e \in E(P)} \ell(e) \leq d$, and $\sum_{e \in E(P)} w(e) \leq \Gamma$.

The following theorem provides an algorithm for the DAG-like SSSP problem. Its formal proof, presented in [CK24], follows almost immediately from the results of [BGWN20]. Specifically, Theorem 5.1 of [BGWN20] provides a similar result with a slightly different setting of parameters, but their arguments can be easily extended to prove the theorem below, as was shown in [CK24].

Theorem 2.7 (Theorem 5.1 in the full version of [CK24]) *There is a deterministic algorithm for the* DAG-like SSSP *problem with total update time $\tilde{O}\left(\frac{n^2+m+\Gamma \cdot n}{\epsilon^2}\right)$; here, m and n are the total number of edges and vertices, respectively, that were ever present in G.*

3 High-Level Overview of the Algorithm

In this section we provide the definitions of well-structured graphs and well-structured cuts, that are used throughout the paper. We also define a new problem, called RouteAndCut, which is one of the main building blocks of our algorithm. We then state the main theorem for this section, that provides an efficient algorithm for the RouteAndCut problem, and we show that our main result – the proof of Theorem 1.1 follows from it. The proof of the theorem that provides an algorithm for the RouteAndCut problem is by induction, and is provided in the remainder of this paper. In this section we also provide a simple and somewhat inefficient algorithm for the RouteAndCut problem, that will be used as the induction base. We also define an intermediate problem, called r-Restricted s-t-SP problem, and show that an algorithm for this problem implies an algorithm for the RouteAndCut problem. We start by defining well-structured graphs and well-structured cuts.

3.1 Well-Structured Graphs and Well-Structured Cuts

Throughout the paper we will work with special kinds of directed graphs, that for convenience we call *well-structured graphs*, and define below.

Definition 3.1 (Well-Structured Graphs) *Let $G = (L, R, E)$ be a bipartite directed graph. We call the edges of $E_G(L, R)$* regular *and the edges of $E_G(R, L)$* special*. We say that G is a* well-structured graph*, if every vertex of G is incident to at most one special edge. If every vertex of G is incident to* **exactly** *one special edge, then we say that G is a* perfect well-structured graph*.*

If G is a well-structured graph, then we assume that the partition (L, R) of its vertices is given as part of the description of G. We will sometimes consider well-structured graphs with lengths on their edges, and we will sometimes restrict our attention to assignments of lengths to edges that have special properties, that we define below. We call such assignments of lengths to edges *proper*.

Definition 3.2 (Proper Assignment of Edge Lengths) *Let $G = (L, R, E)$ be a well-structured directed graph, and let $\ell(e) \geq 0$ be an assignment of lengths to the edges $e \in E$. We say that $\{\ell(e)\}_{e\in E}$ is a* proper *assignment of edge lengths, if, for every regular edge $e \in E$, $\ell(e) = 0$, and for every special edge $e \in E$, $\ell(e)$ is an integral power of 2 with $\ell(e) \geq 1$. If, for every special edge $e \in E$, $\ell(e) = 1$ holds, then we say that $\{\ell(e)\}_{e\in E}$ is a* proper uniform assignment of edge lengths*.*

Recall that a *cut* in a graph G is an ordered pair (A, B) of non-empty subsets of $V(G)$ with $A \cap B = \emptyset$ and $A \cup B = V$. We will often use the notion of a well-structured cut, that is defined next.

Definition 3.3 (Well-Structured Cut) *Let $G = (L, R, E)$ be a well-structured graph, and let (A, B) be a cut in G. We say that (A, B) is a* weakly well-structured cut*, or just a* well-structured cut*, if all edges of $E_G(A, B)$ are special edges. If, additionally, all edges in $E_G(B, A)$ are regular edges, then we say that (A, B) is a* strongly *well-structured cut.*

3.2 The RouteAndCut problem

In this subsection we define the RouteAndCut problem, which is one of the main building blocks of our algorithm. Before we do so, we need to define the notion of *routing*.

Definition 3.4 (Routing) *Let $G = (V, E)$ be a directed graph, and let A, B be two disjoint subsets of its vertices. A* partial routing*, or just a* routing*, from A to B is a collection $\mathcal{Q}$ of paths in G, such* *that every path in $\mathcal{Q}$ connects a vertex of A to a vertex of B, and the endpoints of the paths in $\mathcal{Q}$ are all disjoint. The* congestion *of the routing is the edge-congestion that the set $\mathcal{Q}$ of paths causes in graph G. Vertices of $A \cup B$ may serve both as endpoints and as inner vertices of the paths in $\mathcal{Q}$ simultaneously. A partial routing of cardinality $|A|$ is called a* complete *routing.*

We are now ready to define the RouteAndCut problem, and its special case, called r-restricted RouteAndCut. Intuitively, we use the notion of the r-restricted RouteAndCut problem in order to discretize the problem instances: our algorithm for RouteAndCut will consider, by induction, r-restricted instances of the problem, from smaller to larger values of r.

Definition 3.5 (RouteAndCut problem) *The input to the* RouteAndCut *problem is a well-structured n-vertex graph $G = (L, R, E)$, that is given in the adjacency-list representation, parameters $N \geq n$, $\Delta \geq 1$, and $1 \leq \eta \leq \Delta$, and two disjoint subsets A, B of vertices of G, with $|A|, |B| \geq \Delta$.*

The output of the problem is a (possibly partial) routing $\mathcal{Q}$ from A to B, whose congestion is bounded by $4\eta \log N$. Additionally, if $|\mathcal{Q}| < \Delta$, the output must contain a cut (X, Y) in G with $|E_G(X, Y)| \leq \frac{64\Delta}{\eta \log^4 n} + \frac{256|\mathcal{Q}|}{\eta}$, such that, if $A' \subseteq A, B' \subseteq B$ denote the subsets of vertices that do not serve as endpoints of the paths in $\mathcal{Q}$, then $A' \subseteq X$ and $B' \subseteq Y$ hold. The algorithm for the RouteAndCut *problem is also allowed to return "FAIL" without producing any output, but the probability of the algorithm doing so must be bounded by $1/2$.*

We say that an instance $(G, A, B, \Delta, \eta, N)$ of the RouteAndCut *problem is r-*restricted*, for an integer $1 \leq r \leq \left\lceil \sqrt{\log N} \right\rceil$, iff $\frac{(n-|B|)\cdot\eta}{\Delta} \leq 2^{r\cdot\sqrt{\log N}}$ holds.*

We provide a brief intuition for the parameters Δ and N in the definition of the RouteAndCut problem, and for the definition of r-restricted instances. Consider an arbitrary well-structured n-vertex graph G, two disjoint sets A, B of its vertices and a parameter η, and assume that we need to compute a large routing from A to B in G, with congestion at most η. Typically, we will set Δ to be the cardinality of the desired routing (and if it is not known, we can set $\Delta = \min\{|A|, |B|\}$). Initially, we will also set $N = n$, and consider the resulting instance $(G, A, B, \Delta, \eta, N)$ of the RouteAndCut problem. Now suppose we compute a routing from A to B in graph G of cardinality Δ that causes edge-congestion at most η. Assume w.l.o.g. that, if $b \in B$ is an inner vertex on some path of the routing, then it serves as an endpoint of some other path of the routing. Then it is not hard to show that most of the paths in the routing must have length at most $d = O\left(\frac{(n-|B|)\cdot\eta}{\Delta}\right)$. The definition or r-restricted instances requires that this parameter d is roughly bounded by $2^{r\sqrt{\log N}}$. It is easy to see that our starting instance is r^*-restricted, for $r^* = \left\lceil \sqrt{\log N} \right\rceil$. In order to solve this problem instance, we will need to solve the problem recursively on smaller subgraphs G' of G, but it is important for us to ensure that the resulting instances are r'-restricted, for $r' < r^*$. In order to do so, we will let the parameter N in the resulting subinstances correspond to the number of vertices in the original graph G (we may need to slightly adjust it for technical reasons but the adjustments are minor). By appropriately setting the parameters Δ and η for the resulting subinstances, we can then ensure that they are indeed r'-restricted, for some $r' < r^*$. Overall, the parameter N can be thought of as roughly the number of vertices in the original instance of the RouteAndCut problem that we try to solve, and it is used mostly to define the notion of r-restricted instances. The notion of r-restricted instances allows us to construct algorithms for the RouteAndCut problem inductively, by going from smaller to larger values of r.

The following theorem provides one of our main technical results, namely an efficient algorithm for the RouteAndCut problem.

Theorem 3.6 *There is a randomized algorithm for the* RouteAndCut *problem, that, on an input $(G, A, B, \Delta, \eta, N)$, has running time at most $O\left(n \cdot (n - |B|) \cdot 2^{O(\sqrt{\log N} \cdot \log\log N)}\right)$, where $n = |V(G)|$.*

The remainder of this paper is largely dedicated to the proof of Theorem 3.6, but first we show that the proof of Theorem 1.1 follows from it.

3.3 Completing the Proof of Theorem 1.1

Recall that in the Maximum Bipartite Matching problem, the input is a simple undirected n-vertex bipartite graph $G = (L, R, E)$, and the goal is to compute a matching of maximum cardinality in G.

We can assume w.l.o.g. that we are given a target integral value $C^* > 0$. If $C^* \leq \mathsf{OPT}$, then our algorithm must either produce a matching of cardinality C^*, or terminate with a "FAIL", but we require that the algorithm terminates with a "FAIL" with probability at most $1/\operatorname{poly}(n)$ in this case. If $C^* > \mathsf{OPT}$, then our algorithm may return an arbitrary matching, or terminate with a "FAIL".

We can make this assumption w.l.o.g. since we can use a standard technique of performing a binary search over the value OPT of the optimal solution. We use bounds $W < W'$ on the value of the optimal solution, initializing $W = 1$ and $W' = n$, and we start with $C^* = \lfloor n/2 \rfloor$. Given the current guess C^* on OPT, we run the algorithm with this target value C^*. If the algorithm successfully computes a matching of cardinality at least C^*, then we set $W = C^*$, and otherwise, we set $W' = C^*$. In either case, we select the next value $C^* = \left\lfloor \frac{W'-W}{2} \right\rfloor$ and continue. Once $W' - W \leq 1$ holds, we run the algorithm once with $C^* = W'$ and once with $C^* = W$, and then output the better of the two solutions. We say that an algorithm *errs* in an iteration, if it returns "FAIL", but $C^* \leq \mathsf{OPT}$ holds in that iterations. Notice that, if the algorithm never errs, then after $O(\log n)$ iterations it is guaranteed to terminate with an optimal matching. Since the probability that the algorithm errs in a single iteration is bounded by $1/\operatorname{poly}(n)$, and the number of iterations is bounded by $O(\log n)$, the algorithm is guaranteed to return an optimal matching with probability at least $1 - 1/\operatorname{poly}(n)$.

From now on, we assume that we are given a target value C^*, and that G contains a matching of cardinality C^*. Our algorithm must either compute a matching of cardinality C^*, or to return "FAIL", but the probability for returning "FAIL" must be bounded by $1/\operatorname{poly}(n)$. For convenience, we denote C^* by OPT. We can assume that n is greater than a large enough constant, since otherwise the problem can be solved in $O(1)$ time.

It is well known that the Maximum Bipartite Matching problem can be reduced to computing maximum s-t flow in a directed flow network with unit edge capacities. In order to do so, we start with the graph G, and direct all its edges from the vertices of L towards the vertices of R. We then add a source vertex s, that connects with an edge to every vertex of L, and a destination vertex t, to which every vertex of R is connected. All edge capacities are set to 1. Let G' denote the resulting directed flow network. It is easy to verify that the value of the maximum s-t flow in G' is equal to the cardinality of maximum matching in G. Moreover, given an integral flow f in G', we can compute a matching M of cardinality $\operatorname{val}(f)$ in G, in time $O(n)$: we simply include in M all edges of G that carry 1 flow unit.

Our algorithm consists of $O(\log^3 n)$ phases. Throughout the algorithm, we maintain a matching M in G, starting with $M = \emptyset$, and we denote $\Delta^* = \mathsf{OPT} - |M|$. If M is the matching at the beginning of a phase, and M' is the matching obtained at the end of the phase, then we require that $|M'| \geq |M| + \Omega(\Delta^*/\log^2 n)$, where $\Delta^* = \mathsf{OPT} - |M|$. We show a combinatorial algorithm, that, given an initial matching M, either returns "FAIL", or computes such a matching M', in time $O\left(n^2 \cdot 2^{O(\sqrt{\log n} \log\log n)}\right)$, where the probability that the algorithm returns "FAIL" is bounded by $1/\operatorname{poly}(n)$. This will ensure that the number of the phases is bounded by $O(\log^3 n)$, and the total running time of the algorithm is $O\left(n^2 \cdot 2^{O(\sqrt{\log n}\log\log n)}\right)$. From now on we focus on the description of a single phase.

Implementation of a Single Phase. We assume that we are given some matching M in the input graph G, and we denote $\Delta^* = \mathsf{OPT} - |M|$. Our goal is to compute a matching M' of cardinality at least $|M| + \Omega(\Delta^*/\log^2 n)$. As observed already, matching M defines a flow f of value $|M|$ in the directed flow network G' with unit edge capacities. We denote by $H = G'_f$ the corresponding residual flow network, and we say that H is the *residual flow network of* G *with respect to matching* M. Observe that H is a directed flow network with unit edge capacities, and that the value of the maximum s-t flow in H is at least $\mathsf{OPT} - |M| = \Delta^*$.

Next, we will define an instance of the $\mathsf{RouteAndCut}$ problem. Consider first the graph $H' = H \setminus \{s,t\}$. Notice that, for every edge $e = (u,v)$ of G with $u \in L$ and $v \in R$, if $e \in M$, then edge (v,u) is present in H', and otherwise edge (u,v) is present in H'. Therefore, all edges of H' that are directed from vertices of R to vertices of L correspond to the edges of the current matching M. Clearly, every vertex of H' may be incident to at most one edge of $E_{H'}(R,L)$, and so graph H' is a well-structured graph.

We let $A \subseteq L$ be the set of vertices v, such that edge (s,v) is present in graph H, and we let $B \subseteq R$ be the set of vertices u, such that edge (u,t) is present in H. We also define parameters $\eta = 1$, $N = n$, and $\Delta = \Delta^*$. It is easy to verify that $(H', A, B, \Delta, \eta, N)$ is a valid input to the $\mathsf{RouteAndCut}$ problem, and we can compute an adjacency-list representation of H' in time $O(|E(G)|)$. Recall that there is an s-t flow f of value Δ^* in graph H, and, from the integrality of maximum flow in integer-capacity networks, we can assume that this flow is integral. This flow naturally defines a routing $\mathcal{Q}^*$ from A to B, that causes congestion $\eta = 1$, with $|\mathcal{Q}^*| = \Delta^*$.

We apply the algorithm from Theorem 3.6 to instance $(H', A, B, \Delta, \eta, N)$ of the $\mathsf{RouteAndCut}$ problem. As long as the algorithm returns "FAIL", we keep executing it, for up to $\lceil 100 \log n \rceil$ iterations. If the algorithm from Theorem 3.6 returned "FAIL" in all $\lceil 100 \log n \rceil$ consecutive iterations, we terminate our algorithm and return "FAIL". It is easy to verify that this may happen with probability at most $1/n^{100}$. Otherwise, the algorithm from Theorem 3.6 must return a routing $\mathcal{Q}$ from A to B with congestion at most $4 \log n$. We show that $|\mathcal{Q}| \geq \frac{\Delta^*}{\log n}$ must hold in the next observation.

Observation 3.7 *If $\mathcal{Q}$ is the routing returned by the algorithm from Theorem 3.6, then $|\mathcal{Q}| \geq \frac{\Delta^*}{\log n}$.*

Proof: Let $A' \subseteq A, B' \subseteq B$ denote the subsets of vertices that do not serve as endpoints of the paths in $\mathcal{Q}$. Assume for contradiction that $|\mathcal{Q}| < \frac{\Delta^*}{\log n}$.

Recall that the algorithm from Theorem 3.6 must also return a cut (X,Y) in H' with $|E_{H'}(X,Y)| \leq \frac{64\Delta^*}{\log^4 n} + 256|\mathcal{Q}| \leq \frac{512\Delta^*}{\log n}$, such that $A' \subseteq X$ and $B' \subseteq Y$. Also, recall that we have established that there is a routing $\mathcal{Q}^*$ from A to B in H' of cardinality at least Δ^*, that causes edge-congestion 1. Since $|\mathcal{Q}| < \frac{\Delta^*}{\log n}$, at least $\frac{\Delta^*}{2}$ of the paths in $\mathcal{Q}^*$ must originate at vertices of A' and terminate at vertices of B'. Each such path must contain an edge of $E_{H'}(X,Y)$, so $|E_{H'}(X,Y)| \geq \frac{\Delta^*}{2}$ must hold, a contradiction. □

Notice that the paths in $\mathcal{Q}$ naturally define a collection $\mathcal{Q}'$ of at least $\frac{\Delta^*}{\log n}$ s-t paths in graph H (the residual flow network with respect to G and the current matching M), and they cause congestion at most $4 \log n$ (since the endpoints of the paths in $\mathcal{Q}$ are disjoint). Next, we show an algorithm that computes a collection $\mathcal{Q}''$ of $\Omega\left(\frac{\Delta^*}{\log^2 n}\right)$ edge-disjoint s-t paths in graph H. We will then use the paths in $\mathcal{Q}''$ in order to augment the current flow f in graph G', which, in turn, will allow us to compute the new augmented matching M.

In order to compute the collection $\mathcal{Q}''$ of paths, we construct a directed graph $H'' \subseteq H$, that consists of all vertices and edges that participate in the paths of $\mathcal{Q}'$. The capacity of every edge in H'' remains unit – the same as its capacity in H. Next, we show that $|E(H'')| \leq O(n \log n)$, and that there is an s-t flow of value at least $\frac{\Delta^*}{4\log^2 n}$ in H''.

Observation 3.8 *$|E(H'')| \leq O(n \log n)$, and there is an s-t flow of value at least $\frac{\Delta^*}{4\log^2 n}$ in H''.*

Proof: We start by showing that every vertex $v \in V(H'') \setminus \{s,t\}$ participates in at most $8 \log n$ paths in $\mathcal{Q}'$. Indeed, consider any vertex $v \in V(H'') \setminus \{s,t\}$, and let $Q \in \mathcal{Q}'$ be a path in which vertex v participates. Assume first that $v \in L$. Then either edge (s,v) belongs to H and lies on path Q, or there is a special edge (u,v) incident to v in H, that lies on path Q. Since the paths in $\mathcal{Q}'$ cause congestion at most $4 \log n$ in H, there may be at most $8 \log n$ paths in $\mathcal{Q}'$ that contain either of the edges (s,v) or (u,v), and so v participates in at most $8 \log n$ paths in $\mathcal{Q}'$. Using a similar argument, every vertex $u \in R$ participates in at most $8 \log n$ paths in $\mathcal{Q}'$. Therefore, $|E(H'')| \leq O(n \log n)$.

Consider now a flow in H'', where we send $\frac{1}{4\log n}$ flow units along every path in $\mathcal{Q}'$. Since the paths in $\mathcal{Q}'$ cause congestion at most $4 \log n$, this is a valid s-t flow in H'', and the value of this flow is $\frac{|\mathcal{Q}'|}{4\log n} \geq \frac{\Delta^*}{4\log^2 n}$. □

Next, we compute an integral maximum s-t flow in H'' that obeys the edge capacities in H'', using the standard Ford-Fulkerson algorithm. We start by setting the flow $f'(e)$ on every edge $e \in E(H')$ to 0, and then perform iterations. In every iteration, we use the current flow f' in H'' in order to compute a residual flow network $H''_{f'}$. If there is no s-t path in $H''_{f'}$, then the algorithm terminates. We are then guaranteed that the value of the current flow f' is optimal, so $\text{val}(f') \geq \frac{\Delta^*}{4\log^2 n}$. Otherwise, we compute an arbitrary s-t path P in $H''_{f'}$, and we augment the flow f' along the path P. Notice that every iteration of the algorithm can be implemented in time $O(|E(H''_{f'})|) = \widetilde{O}(n)$, and the number of iterations is bounded by n. Therefore, the total running time of the algorithm that computes the set $\mathcal{Q}''$ of paths in H'' is $\widetilde{O}(n^2)$. The final integral flow f' can be decomposed, in time $O(|E(H'')|) = \widetilde{O}(n)$, into a collection $\mathcal{Q}''$ of at least $\Omega\left(\frac{\Delta^*}{\log^2 n}\right)$ edge-disjoint paths connecting s to t in H''. Since $H'' \subseteq H$, we have now obtained the desired collection $\mathcal{Q}''$ of at least $\Omega\left(\frac{\Delta^*}{\log^2 n}\right)$ edge-disjoint s-t paths in H.

We note that we could also directly round the initial fractional s-t flow in H'', in expected time $\widetilde{O}(|E(H'')|) = \widetilde{O}(n)$, e.g. by using the algorithm from Theorem 5 in [LRS13], that builds on the results of [GKK10]. But since their running time is only bounded in expectation, and since the bottlenecks in the running time of our algorithm lie elsewhere, we instead use the above simple deterministic algorithm.

We can now augment the current flow in G' via the collection $\mathcal{Q}''$ of augmenting paths, obtaining a new integral flow in graph G' of value $|M| + |\mathcal{Q}''|$, which, in turn, defines a new matching M' with $|M'| \geq |M| + |\mathcal{Q}''| \geq |M| + \Omega\left(\frac{\Delta^*}{\log^2 n}\right)$.

We now bound the running time of a single phase. Recall that we may execute the algorithm from Theorem 3.6 at most $O(\log n)$ times per phase. The running time of a single execution of the algorithm from Theorem 3.6 is bounded by $O\left(n^2 \cdot 2^{O(\sqrt{\log N} \cdot \log\log N)}\right) \leq O\left(n^2 \cdot 2^{O(\sqrt{\log n} \cdot \log\log n)}\right)$. Additionally, the time required to compute the graph H'', and to compute the maximum flow in it is bounded by $\widetilde{O}(n^2)$.

Overall, the running time of a single phase is bounded by $O\left(n^2 \cdot 2^{O(\sqrt{\log n} \cdot \log\log n)}\right)$. Since the number of phases is bounded by $O(\log^3 n)$, the total running time of the algorithm is bounded by

$O\left(n^2 \cdot 2^{O(\sqrt{\log n}\cdot \log\log n)}\right)$, and the probability that the algorithm ever returns "FAIL" is bounded by $1/\operatorname{poly}(n)$.

3.4 Proof of Theorem 3.6 – High Level Overview

In order to prove Theorem 3.6, we use two large enough constants $c_1 \gg c_2$, whose values we set later, and we employ the following theorem.

Theorem 3.9 *For all $r \geq 1$, there is a randomized algorithm for the r-restricted* RouteAndCut *problem, that, on an input $(G, A, B, \Delta, \eta, N)$ with $|V(G)| = n$, has running time at most:*

$$c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2}.$$

Note that, if $(G, A, B, \Delta, \eta, N)$ is an instance of the RouteAndCut problem, and $|V(G)| = n$, then, from the problem definition, $\eta \leq \Delta$ and $\frac{(n-|B|)\cdot\eta}{\Delta} \leq n$ holds. Therefore, any instance of the problem is also an instance of r^*-restricted RouteAndCut problem, for $r^* = \left\lceil\sqrt{\log n}\right\rceil$. By using Theorem 3.9 with parameter $r^* = \left\lceil\sqrt{\log n}\right\rceil$, we obtain an algorithm for the general RouteAndCut problem whose running time is:

$$O\left(n \cdot (n - |B|) \cdot 2^{O(\sqrt{\log N}\cdot \log\log N)}\right).$$

Therefore, in order to prove Theorem 3.6, it is enough to prove Theorem 3.9. In the remainder of this paper we focus on the proof of Theorem 3.9. The proof is by induction on r, and it relies on a slight modification of the Multiplicative Weight Update (MWU) framework of [GK98, Fle00], that essentially reduces the r-restricted RouteAndCut problem to a special case of the directed SSSP problem, that we call r-Restricted s-t-SP. Before we describe this approach, we describe an algorithm that, given an instance $(G, A, B, \Delta, \eta, N)$ of the RouteAndCut problem, constructs a subgraph $G' \subseteq G$ with $V(G') = V(G)$ and $|E(G')| \leq O((n - |B|) \cdot n)$, such that a solution to the RouteAndCut problem instance $(G', A, B, \Delta, \eta, N)$ immediately provides a solution to the RouteAndCut problem instance $(G, A, B, \Delta, \eta, N)$.

3.4.1 An Auxiliary Graph G' and its Properties

Consider an instance $(G = (L, R, E), A, B, \Delta, \eta, N)$ of the r-restricted RouteAndCut problem, for an integer $1 \leq r \leq \left\lceil\sqrt{\log N}\right\rceil$. Let $G' \subseteq G$ be a graph, that is constructed as follows. We let $V(G') = V(G)$, and we include in G' all special edges of G. Additionally, for every vertex $v \in L$, if $|\delta^+_G(v)| \leq 512(n - |B|)$, then we add all regular edges that leave v in G to graph G', and otherwise, we select an arbitrary subset $E'(v) \subseteq \delta^+_G(v)$ of $512(n - |B|)$ such edges, and include these edges in G'.

Note that G' is a well-structured n-vertex graph, and $(G', A, B, \Delta, \eta, N)$ is a valid instance of the r-restricted RouteAndCut problem. Moreover, the out-degree of every vertex $v \in L$ is bounded by $512(n - |B|)$, and so $|E(G')| \leq O((n - |B|) \cdot n)$. Given the adjacency-list representation of graph G, graph G' can be constructed explicitly in time $O((n - |B|) \cdot n)$.

Since $G' \subseteq G$, any routing $\mathcal{Q}$ from A to B with congestion at most $4\eta\log N$ in G' defines a routing from A to B with congestion at most $4\eta \log N$ in G. Assume now that we are given a routing $\mathcal{Q}$ from A to B in G' with congestion at most $4\eta\log N$, such that $|\mathcal{Q}| < \Delta$. Let $A' \subseteq A, B' \subseteq B$ denote the subsets of vertices that do not serve as endpoints of the paths in $\mathcal{Q}$. Assume, additionally, that we are given a cut (X,Y) in G' with $|E_{G'}(X,Y)| \leq \frac{64\Delta}{\eta\log^4 n} + \frac{256|\mathcal{Q}|}{\eta}$, such that $A' \subseteq X$ and $B' \subseteq Y$. Clearly, (X,Y) is also a cut in graph G. We claim that $E_G(X,Y) = E_{G'}(X,Y)$. Indeed, since $|\mathcal{Q}| < \Delta$, we get that $|B'| > |B| - \Delta$, and so $|X| \leq n - |B'| < n - |B| + \Delta \leq 2(n-|B|)$ (here, we have used the fact that $\Delta \leq |A| \leq (n-|B|)$). Assume for contradiction that $E_G(X,Y) \neq E_{G'}(X,Y)$. Since $G' \subseteq G$, there must be an edge $e = (x,y) \in E_G(X,Y)$, such that $e \notin E(G')$. This can only happen if $x \in L$, and graph G' contains a subset $E'(x) \subseteq \delta^+_G(x)$ of $512(n-|B|)$ edges. Since $|X| \leq 2(n-|B|)$, at least $510(n-|B|)$ edges of $E'(x)$ must lie in $E_{G'}(X,Y)$. Therefore, $|E_{G'}(X,Y)| \geq 510(n-|B|)$ must hold. However, $|E_{G'}(X,Y)| \leq \frac{64\Delta}{\eta\log^4 n} + \frac{256|\mathcal{Q}|}{\eta} \leq 509\Delta \leq 509(n-|B|)$, a contradiction. From the above discussion, we obtain the following claim.

Claim 3.10 *There is a deterministic algorithm, that, given an instance (G,A,B,Δ,η,N) of the r-restricted* RouteAndCut *problem (where graph G is given as an adjacency list), constructs another instance (G',A,B,Δ,η,N) of the r-restricted* RouteAndCut *problem with $G' \subseteq G$, such that $|E(G')| \leq O(n\cdot(n-|B|))$, and moreover, if $(\mathcal{Q},(X,Y))$ is a valid solution to instance (G',A,B,Δ,η,N), then it is also a valid solution to instance (G,A,B,Δ,η,N). The running time of the algorithm is $O(n\cdot(n-|B|))$.*

In our algorithms for the RouteAndCut problem, we will use Claim 3.10 in order to ensure that the number of edges in the input graph is bounded by $O(n\cdot(n-|B|))$. We now turn to describe the MWU-based approach for solving r-restricted RouteAndCut.

3.5 Solving r-Restricted RouteAndCut via the Modified MWU Framework and a Reduction to r-Restricted s-t-SP

In this subsection we provide a high-level description of our algorithm for the r-restricted RouteAndCut problem, including the modified MWU framework that we use, and a reduction to a problem that we call r-restricted s-t-SP, that can be viewed as a special case of the SSSP problem. We show that an algorithm for the r-restricted s-t-SP problem implies an algorithm for the r-restricted RouteAndCut problem, via the modified MWU framework. Lastly, we provide a simple though not very efficient algorithm for r-restricted s-t-SP, that will be used as the induction base in the proof of Theorem 3.9.

Let (G,A,B,Δ,η,N) be the input to the r-restricted RouteAndCut problem instance, where $G = (L,R,E)$ is a well-structured graph with $|V(G)| = n$, that is given as an adjancency list, A and B are disjoint subsets of $V(G)$, and $N \geq n$, $\Delta \geq 1$, and $1 \leq \eta \leq \Delta$ are parameters, with $|A|,|B| \geq \Delta$. Since the instance is r-restricted, $\frac{(n-|B|)\cdot\eta}{\Delta} \leq 2^{r\cdot\sqrt{\log N}}$ holds. By using the algorithm from Claim 3.10, we convert this instance into another instance (G',A,B,Δ,η,N) of r-restricted RouteAndCut with $|E(G')| \leq O(n\cdot(n-|B|))$, in time $O(n\cdot(n-|B|))$. From now on we focus on solving instance (G',A,B,Δ,η,N), and, for convenience, we denote G' by G.

Recall that our goal is to compute a routing $\mathcal{Q}$ from A to B, whose congestion is bounded by $4\eta\log N$. Additionally, if $|\mathcal{Q}| < \Delta$, we need to compute a cut (X,Y) in G with $|E_G(X,Y)| \leq \frac{64\Delta}{\eta\log^4 n} + \frac{256|\mathcal{Q}|}{\eta}$, such that, if $A' \subseteq A, B' \subseteq B$ denote the subsets of vertices that do not serve as endpoints of the paths in $\mathcal{Q}$, then $A' \subseteq X$ and $B' \subseteq Y$ hold. Recall also that the edges of $E_G(R,L)$ are called special edges, and the remaining edges of G are called regular edges. From the definition of well-structured graphs, every vertex of G is incident to at most one special edge. We start with a simple preprocessing step.

A Preprocessing Step. As a preprocessing step, we construct a maximal collection $\mathcal{Q}_0$ of disjoint paths, where every path connects a distinct vertex of A to a distinct vertex of B, and consists of a single regular edge. We do so by employing a simple greedy algorithm: Start with $\mathcal{Q}_0 = \emptyset$. While there is a regular edge $e = (a,b)$ with $a \in A$ and $b \in B$, add a path $Q = (e)$ to $\mathcal{Q}_0$, and delete a and b from A and

B, respectively. Clearly, this preprocessing step can be executed in time $O(|E(G)|) \leq O(n \cdot (n - |B|))$. If $|\mathcal{Q}_0| \geq \Delta$, then we terminate the algorithm and return $\mathcal{Q}_0$. We assume from now on that $|\mathcal{Q}_0| < \Delta$. Let $A_1 \subseteq A$, $B_1 \subseteq B$ denote the sets of vertices that do not serve as endpoints of the paths in $\mathcal{Q}_0$. Since $|\mathcal{Q}_0| < \Delta$, we get that $|B_1| \geq |B| - \Delta$, and so:

$$n - |B_1| \leq n - |B| + \Delta \leq 2(n - |B|), \tag{1}$$

since $n - |B| \geq |A| \geq \Delta$ from the problem definition. The preprocessing step ensures that there is no regular edge in G connecting a vertex of A_1 to a vertex of B_1, so every path connecting a vertex of A_1 to a vertex of B_1 must contain at least one special edge.

Next, we describe a modified MWU framework that will allow us to compute a solution to the r-restricted RouteAndCut problem instance. The modified MWU framework is designed in such a way that it reduces the r-restricted RouteAndCut problem to a special case of SSSP, that we call r-restricted s-t-SP, which seems more tractable than the general decremental SSSP problem in directed graphs. We will then design an algorithm for the r-restricted s-t-SP problem, that will allow us to obtain the desired algorithm for r-restricted RouteAndCut. The algorithm for r-restricted s-t-SP will in turn rely on an algorithm for the $(r-1)$-restricted RouteAndCut problem.

3.5.1 The Modified MWU Framework

We now describe an algorithm that is based on the modified MWU framework, but ignore the issue of the efficient implementation of the algorithm. We address this issue later, by reducing the problem to the r-restricted s-t-SP problem.

Throughout the algorithm, we use a parameter $\Lambda = (n - |B_1|) \cdot \frac{\eta \log^5 n}{\Delta}$. While the set B_1 of vertices may change over the course of the algorithm, the value of the parameter Λ is set at the beginning of the algorithm and remains unchanged throughout the algorithm. Our algorithm maintains an assignment $\ell(e) \geq 0$ of *lengths* to the edges of G. At the beginning of the algorithm, we assign an initial length $\ell(e)$ to every edge $e \in E(G)$, as follows. If e is a regular edge, we set $\ell(e) = 0$, and if it is a special edge, we set $\ell(e) = \frac{1}{\Lambda}$.

As the algorithm progresses, the lengths of the special edges may grow, but the lengths of the regular edges remain unchanged. Whenever we talk about distances between vertices and lengths of paths, it is always with respect to the current lengths $\ell(e)$ of edges $e \in E(G)$.

The algorithm gradually constructs a routing $\mathcal{Q}$ from A_1 to B_1 in G, starting with $\mathcal{Q} = \emptyset$. It then performs at most Δ iterations, and in every iteration, a single path P, connecting some vertex $a \in A_1$ to some vertex $b \in B_1$, is added to $\mathcal{Q}$. We add P to $\mathcal{Q}$, and delete a from A_1 and b from B_1. Additionally, we may double the lengths of *some* special edges on the path P. Specifically, our algorithm maintains, for every special edge e of G, a counter $n(e)$, which, intuitively, counts the number of paths that were added to $\mathcal{Q}$ that contained e, since $\ell(e)$ was last doubled (or since the beginning of the algorithm, if $\ell(e)$ was never doubled yet). At the beginning of the algorithm, we set $n(e) = 0$ for every special edge e. Whenever a path P is added to $\mathcal{Q}$, we increase the counter $n(e)$ of every special edge $e \in E(P)$. If, for any such edge e, the counter $n(e)$ reaches η, then we double the length of edge e, and reset the counter $n(e)$ to be 0.

The Oracle

At the heart of our algorithm is an *oracle* – an algorithm that, in every iteration, either computes a path P of length at most 1 in the current graph G connecting a vertex of A_1 to a vertex of B_1, or produces a new assignment $\ell'(e)$ for edges $e \in E(G)$, that we call a *cut-witness*, and define next. Intuitively, the cut-witness is designed in such a way that it can be easily transformed into the desired cut (X, Y) in G with $|E_G(X,Y)| \leq \frac{64\Delta}{\eta \log^4 n} + \frac{256|\mathcal{Q}|}{\eta}$, such that, if A' and B' denote the current sets A_1 and B_1 respectively, then $A' \subseteq X$ and $B' \subseteq Y$ hold. Once the oracle produces a cut-witness, the algorithm terminates. We now define the cut-witness formally.

Definition 3.11 (Cut-witness) *A* cut-witness *is an assignment of lengths $\ell'(e) \geq 0$ to every edge $e \in E(G)$, such that, if we denote by A' and B' the current sets A_1 and B_1 of vertices respectively, and by E^* the set of all special edges with both endpoints in B', then:*

- $\sum_{e \in E(G)} \ell'(e) \leq \frac{\Delta}{2\eta \log^4 n} + \sum_{e \in E(G) \setminus E^*} \ell(e)$*; and*
- *the distance in graph G, with respect to edge lengths $\ell'(\cdot)$, from A' to B' is at least $\frac{1}{64}$.*

Next, we define the notion of an *acceptable path* in G, and we will require that the oracle, in every iteration, either returns an acceptable path, or returns a cut-witness.

Definition 3.12 (Acceptable Path) *A path P in the current graph G is called* acceptable *if P is a simple path, connecting a vertex of A_1 to a vertex of B_1, the length of P with respect to the current edge lengths $\ell(\cdot)$ is at most 1, and no inner vertices of P belong to B_1.*

We are now ready to define the oracle.

Definition 3.13 (The Oracle) *An oracle for the* MWU *framework is an algorithm that, in every iteration, either returns an acceptable path P, or returns a cut-witness, or returns "FAIL". The probability that the oracle ever returns "FAIL" must be bounded by 1/2.*

The MWU-Based Algorithm.

We describe the modified MWU-based algorithm, denoted by ALG-MWU in Figure 2; the description excludes the implementation of the oracle, that will be discussed later.

Before we discuss the implementation of the oracle, and an efficient implementation of Algorithm ALG-MWU, we first show that we can use the algorithm in order to solve the r-restricted RouteAndCut problem. Observe first that, if the total running time of the oracle, over the course of all iterations, is bounded by T, then the total running time of Algorithm ALG-MWU is bounded by $O(|T|) + O(|E(G)|) \leq O(|T|) + O(n \cdot (n - |B|))$. Moreover, since the probability that the oracle ever returns "FAIL" is at most 1/2, the probability that algorithm ALG-MWU terminates with a "FAIL" is bounded by 1/2.

Let $\mathcal{Q}$ be the set of paths obtained when Algorithm ALG-MWU terminates, and let $\mathcal{Q}' = \mathcal{Q} \cup \mathcal{Q}_0$ be the final set of paths that we obtain. Clearly, every path in $\mathcal{Q}'$ connects a vertex of A to a vertex of B. We start by showing that the paths in $\mathcal{Q}'$ cause congestion at most $4\eta \log n$, and that the endpoints of all paths in $\mathcal{Q}'$ are disjoint, in the following simple observation.

Observation 3.14 *The endpoints of the paths in $\mathcal{Q}'$ are disjoint, and the congestion caused by the paths in $\mathcal{Q}'$ in G is bounded by $4\eta \log n$.*

Alg-MWU

- Let $A_1 \subseteq A$ and $B_1 \subseteq B$ be the sets of vertices that do not serve as endpoints of any path in the set $\mathcal{Q}_0$ constructed in the preprocessing step.
- Initialize the data structures:
  - Set $\mathcal{Q} = \emptyset$;
  - For every edge $e \in E(G)$, if e is a regular edge, set $\ell(e) = 0$, otherwise set $\ell(e) = \frac{1}{\Lambda}$.
  - For every special edge e, set $n(e) = 0$.
- Perform at most $\Delta - |\mathcal{Q}_0|$ iterations, where in each iteration we apply the oracle.
  - If the oracle returned "FAIL", then return "FAIL" and terminate the algorithm;
  - If the oracle returned a cut-witness, return the cut-witness and terminate the algorithm;
  - Otherwise, the oracle must have returned an acceptable path P connecting a vertex $a \in A_1$ to a vertex $b \in B_1$.
    - remove a from A_1 and b from B_1;
    - add P to $\mathcal{Q}$;
    - for each special edge $e \in E(P)$, increase $n(e)$ by 1, and, if $n(e)$ reaches η, double $\ell(e)$ and set $n(e) = 0$.

Figure 2: Alg-MWU

Proof: It is easy to verify that the endpoints of all paths in $\mathcal{Q}'$ are disjoint. First, the preprocessing step, in every iteration selects a regular edge $e = (a, b)$ with $a \in A$ and $b \in B$, adds the path $Q = (e)$ to Q_1, and then deletes a from A and b from B. This ensures that vertices a and b cannot serve as endpoints of any paths that are subsequently added to $\mathcal{Q}_0$, or to $\mathcal{Q}$. Additionally, whenever Algorithm ALG-MWU adds a new path P connecting a vertex $a' \in A_1$ to a vertex $b' \in B_1$ to $\mathcal{Q}$, we delete a' from A_1 and b' from B_1. Therefore, a' and b' may not serve as endpoints of paths that are added subsequently to $\mathcal{Q}$. Hence the endpoints of all paths in $\mathcal{Q}'$ are disjoint.

Next, we show that every *special edge* of G belongs to at most $2\eta \log n$ paths of $\mathcal{Q}$. At the beginning of the algorithm, we set $\ell(e) = \frac{1}{\Lambda} = \frac{\Delta}{(n-|B_1|)\cdot\eta\cdot\log^5 n} \geq \frac{\Delta}{n\cdot\eta\cdot\log^5 n}$. Let $i_1, i_2, \ldots, i_q$ denote the indices of the iterations when $\ell(e)$ was doubled. Notice that, once $\ell(e)$ grows above 1, it may no longer be used by any paths that are added to $\mathcal{P}$, so $q \leq \lceil \log \Lambda \rceil + 1 \leq 2 \log n$ must hold. Moreover, for all $1 \leq j < q$, the number of paths that were added to $\mathcal{Q}$ after iteration i_j but until iteration i_{j+1} (inclusive), that contained e, is bounded by η. Similarly, at most η paths containing e were added to $\mathcal{Q}$ until iteration i_1 (inclusive). Altogether, e may belong to at most $2\eta \log n$ paths of $\mathcal{Q}$. Since paths of $\mathcal{Q}_0$ do not contain special edges, *every special edge belongs to at most* $2\eta \log n$ *paths in* $\mathcal{Q}'$.

Consider now some *regular edge* $e = (u, v) \in E(G)$. Recall that no regular edge in G may connect a vertex of A_1 to a vertex of B_1. Therefore, if a path $P \in \mathcal{Q}$ contains e, then it must contain either the unique special edge that is incident to u (if it exists), or the unique special edge that is incident to v (if it exists). Moreover, if a path $Q = (e)$ belongs to $\mathcal{Q}_0$, then every path $Q' \in \mathcal{Q}$ that contains e must contain both the unique special edge incident to u and the unique special edge incident to v (since $u \notin A_1$ and $v \notin B_1$ must hold). Therefore, every regular edge e belongs to at most $4\eta \log n$ paths in $\mathcal{Q}'$. □

Let A' and B' denote the sets A_1 and B_1, respectively, at the end of Algorithm ALG-MWU. Notice that A' is a set of all vertices $a \in A$ that do not serve as endpoints of the paths in $\mathcal{Q}'$, and similarly, B' contains all vertices $b \in B$ that do not serve as endpoints of the paths in $\mathcal{Q}'$. We now show an algorithm, that, given a cut-witness $\{\ell'(e)\}_{e \in E(G)}$, computes a cut (X, Y) in G with $|E_G(X, Y)| \leq \frac{64\Delta}{\eta \log^4 n} + \frac{256|\mathcal{Q}|}{\eta}$, such that $A' \subseteq X$ and $B' \subseteq Y$ hold.

Claim 3.15 *There is a deterministic algorithm, that, given a cut-witness* $\{\ell'(e)\}_{e \in E(G)}$ *for* G*, obtained at the end of Algorithm* ALG-MWU*, computes a cut* (X, Y) *in* G *with* $|E_G(X, Y)| \leq \frac{64\Delta}{\eta \log^4 n} + \frac{256|\mathcal{Q}'|}{\eta}$*, such that* $A' \subseteq X$ *and* $B' \subseteq Y$ *hold. The running time of the algorithm is* $O(|E(G)|) \leq O(n \cdot (n - |B|))$*.*

Proof: Consider the cut-witness $\{\ell'(e)\}_{e \in E(G)}$. We start by bounding $\sum_{e \in E(G)} \ell'(e)$ in the following simple observation.

Observation 3.16 $\sum_{e \in E(G)} \ell'(e) \leq \frac{\Delta}{\eta \log^4 n} + \frac{2|\mathcal{Q}'|}{\eta}$.

Proof: For every special edge $e \in E(G)$, let $\ell^0(e)$ denote the length of e at the beginning of Algorithm ALG-MWU, and let $\ell^*(e)$ denote the length of e at the end of the algorithm. We denote by E' the set of all special edges $e \in E(G)$, such that, at the beginning of Algorithm ALG-MWU, both endpoints of E' lied in B_1, but $e \notin E^*$. For every edge $e \in E'$, at least one endpoint of e was deleted from B_1 over the course of the algorithm, and e is the only special edge incident to that endpoint, so $|E'| \leq \Delta$ must hold. Recall that, at the beginning of the algorithm, every special edge was assigned the length $\frac{1}{\Lambda}$. Let E^{spec} denote the set of all special edges in G. For each edge $e \in E^{\mathsf{spec}}$, if $e \notin E' \cup E^*$, then, at the beginning of Algorithm ALG-MWU, e was incident to some vertex $x \in V(G) \setminus B_1$, and moreover, e is the only special edge incident to x. Therefore, $|E(G) \setminus (E' \cup E^*)| \leq n - |B_1|$, and $|E(G) \setminus E^*| \leq n - B_1 + \Delta \leq 2(n - |B_1|)$, since $\Delta \leq |A| \leq n - |B_1|$. Overall, we get that:

$$\sum_{e\in E(G)\setminus E^*}\ell^0(e)\le\frac{2(n-|B_1|)}{\Lambda}\le\frac{2\Delta}{\eta\cdot\log^5 n},$$

since $\Lambda=(n-|B_1|)\cdot\frac{\eta\log^5 n}{\Delta}$.

Let $\hat{E}$ denote the subset of special edge $e\in E^{\mathsf{spec}}\setminus E^*$ with $\ell^*(e)>\ell^0(e)$. Clearly, $\sum_{e\in E(G)\setminus E^*}\ell^*(e)\le\sum_{e\in E(G)\setminus E^*}\ell^0(e)+\sum_{e\in\hat{E}}\ell^*(e)$.

Let q be the total number of iterations Algorithm ALG-MWU, so $|\mathcal{Q}|=q$ holds, and denote $\mathcal{Q}=\{P_1,\ldots,P_q\}$, where the paths are indexed in the order in which they were added to $\mathcal{Q}$. For all $1\le i\le q$, let C_i denote the length of path P_i when it was added to $\mathcal{Q}$. Clearly, $C_i\le 1$, and $\sum_{i=1}^q C_i\le q$.

Consider now some edge $e\in\hat{E}$, and recall that $\ell(e)$ was doubled at least once during the algorithm. Let τ be the last time when the length of e was doubled. Then, prior to time τ, there were at least η iterations i, during which the path P_i that was added to $\mathcal{Q}$ contained e, and the length of e during the corresponding iteration was $\frac{\ell^*(e)}{2}$. In other words, edge e contributes at least $\frac{\ell^*(e)\eta}{2}$ to $\sum_{i=1}^q C_i$. Therefore, we get that:

$$\sum_{e\in\hat{E}}\ell^*(e)\le\frac{2\sum_{i=1}^q C_i}{\eta}\le\frac{2q}{\eta}\le\frac{2|\mathcal{Q}|}{\eta}.$$

Altogether, we get that:

$$\sum_{e\in E(G)\setminus E^*}\ell^*(e)\le\sum_{e\in E(G)\setminus E^*}\ell^0(e)+\sum_{e\in\hat{E}}\ell^*(e)\le\frac{2\Delta}{\eta\cdot\log^5 n}+\frac{2|\mathcal{Q}|}{\eta}.$$

Lastly, from the definition of the cut-witness, we get that:

$$\sum_{e\in E(G)}\ell'(e)\le\frac{\Delta}{2\eta\log^4 n}+\sum_{e\in E(G)\setminus E^*}\ell(e)\le\frac{\Delta}{\eta\log^4 n}+\frac{2|\mathcal{Q}|}{\eta}\le\frac{\Delta}{\eta\log^4 n}+\frac{2|\mathcal{Q}'|}{\eta}.$$

□

Consider the graph G with lengths $\ell'(e)$ on its edges $e\in E(G)$. From the definition of a cut-witness, the length of a shortest-path (with respect to edge-lengths $\ell'(\cdot)$) connecting a vertex of A' to a vertex of B' in G' is at least $\frac{1}{64}$.

Consider a thought experiment, where we select a threshold $\rho\in(0,\frac{1}{64})$ uniformly at random, and then let X be the set of all vertices $v\in V(G)$, such that the distance from A' to v, with respect to the length $\ell'(\cdot)$ of edges, is at most ρ. Let $Y=V(G)\setminus X$. It is easy to verify that, since the distance from A' to B' in G is at least $\frac{1}{64}$, $A'\subseteq X$ and $B'\subseteq Y$ must hold. Moreover, the probability that an edge $e\in E(G)$ lies in $E_G(X,Y)$ is bounded by $64\ell'(e)$. Therefore, the expectation of $|E_G(X,Y)|$ is bounded by $64\sum_{e\in E(G)}\ell'(e)\le\frac{64\Delta}{\eta\log^4 n}+\frac{128|\mathcal{Q}'|}{\eta}$.

By performing a standard ball-growing procedure from the set A' of vertices in graph G, we can compute, in time $O(|E(G)|)$, a cut (X,Y) in G with $|E_G(X,Y)|\le\frac{64\Delta}{\eta\log^4 n}+\frac{256|\mathcal{Q}'|}{\eta}$, such that $A'\subseteq X$ and $B'\subseteq Y$. The total running time of the algorithm is bounded by $O(|E(G)|)\le O(n\cdot(n-|B|))$. □

To summarize, we provided an algorithm for the r-restricted RouteAndCut problem, that assumes the existence of an oracle.

We now discuss an efficient implementation of our algorithm for the RouteAndCut problem, given an efficient implementation of the oracle. Recall that the time required for the preprocessing step, and for computing the final cut (X,Y) by the algorithm from Claim 3.15, is bounded by $O(n\cdot(n-|B|))$. If the running time of the oracle is bounded by T, then the running time of Algorithm ALG-MWU is bounded by $O(T+n\cdot(n-B))$. Therefore, the total running time of the algorithm for the RouteAndCut problem is bounded by $O(T+n\cdot(n-|B|))$. We now focus on implementing the oracle.

Implementing the Oracle: Intuition.

Below we define a problem called r-restricted s-t-SP. The problem is designed to implement the oracle from Definition 3.13. However, for convenience, we multiply all edge lengths by factor Λ, and then round them up to the next integer. We denote the new edge lengths by $\hat{\ell}(e)$ for $e\in E(G)$. Notice that, throughout the algorithm, the length $\hat{\ell}(e)$ of every edge $e\in E(G)$ is either 0, or it is an integer between 1 and $\lceil\Lambda\rceil$. We now define the analogues of an acceptable path and a cut-witness with respect to the new edge lengths, that we refer to as *modified acceptable path* and *modified cut-witness*, respectively.

Definition 3.17 (Modified Acceptable Path) *A path P in the current graph G is called a* modified acceptable path for G with respect to A_1 and B_1, *if P is a simple path, connecting a vertex of A_1 to a vertex of B_1, the length of P with respect to the current edge lengths $\hat{\ell}(\cdot)$ is at most Λ, and no inner vertices of P belong to B_1.*

Definition 3.18 (Modified Cut-witness) *A* modified cut-witness *for graph G with respect to A_1 and B_1 is an assignment of lengths $\ell''(e)\geq 0$ to every edge $e\in E(G)$, such that if we denote by A',B' the current sets A_1,B_1 of vertices, and by E^* the set of all special edges with both endpoints in B', then:*

- $\sum_{e\in E(G)}\ell''(e)\leq\frac{\Lambda\cdot\Delta}{\eta\log^4 n}+\sum_{e\in E(G)\setminus E^*}\hat{\ell}(e)$*; and*
- *the distance in graph G, with respect to edge lengths $\ell''(\cdot)$, from A' to B' is at least $\frac{\Lambda}{32}$.*

It is immediate to verify that, if $\{\ell''(e)\}_{e\in E(G)}$ is a modified cut-witness for G, then, by setting the length $\ell'(e)=\frac{\ell''(e)}{2\Lambda}$ for every edge $e\in E(G)$, we obtain a valid cut-witness for G. Additionally, if P is a valid modified acceptable path for G, then it is also an acceptable path for G.

In order to implement the oracle, it is now enough to design an algorithm that, in every iteration, either produces a modified acceptable path in G, or produces a modified cut-witness for G, or returns "FAIL", such that the probability that the algorithm ever returns "FAIL" is bounded by 1/2. Recall that $n-|B_1|\leq 2(n-|B|)$ from Inequality 1, and so, from the definition of r-restricted RouteAndCut problem, $\frac{(n-|B_1|)\cdot\eta}{\Delta}\leq\frac{2(n-|B|)\cdot\eta}{\Delta}\leq 2\cdot 2^{r\cdot\sqrt{\log N}}$ holds.

We now provide a formal definition of the r-restricted s-t-SP problem, that is designed specifically to provide such an algorithm. For convenience, in the definition of the problem, we will denote the sets A_1,B_1 of vertices by A and B, respectively, and we will denote the edge lengths $\hat{\ell}(e)$ by $\ell(e)$.

3.5.2 The r-Restricted s-t-SP Problem

The input to the r-Restricted s-t-SP problem is a well-structured n-vertex graph $G=(L,R,E)$, parameters $N\geq n$, $\Delta\geq 1$, and $1\leq\eta\leq\Delta$, and two disjoint subsets A,B of vertices of G, such that no regular edge of G connects a vertex of A to a vertex of B in G, $\Delta\leq n-|B|$, and $\frac{(n-|B|)\cdot\eta}{\Delta}\leq 2\cdot 2^{r\cdot\sqrt{\log N}}$ holds. We also require that $|E|\leq O(n\cdot(n-|B|))$ and that $1\leq r\leq\left\lceil\sqrt{\log N}\right\rceil$.

Initially, every edge $e \in E(G)$ is assigned a length $\ell(e)$ as follows: if e is a regular edge then $\ell(e) = 0$, and otherwise $\ell(e) = 1$. We use a parameter $\Lambda = \frac{(n-|B|)\eta \cdot \log^5 n}{\Delta}$, and we initialize $\mathcal{Q} = \emptyset$. While the set B of vertices may change over the course of the algorithm, parameter Λ remains unchanged.

The execution of the algorithm for the r-restricted s-t-SP problem consists of at most Δ iterations. In every iteration i, the algorithm must do one of the following:

- either return FAIL;
- or return a modified cut-witness for G with respect to A and B (see Definition 3.18);
- or return a modified acceptable path P for G with respect to A and B (see Definition 3.17).

In the former two cases, the algorithm terminates. In the last case, let $a \in A$, $b \in B$ denote the endpoints of path P. We remove a from A and b from B and add P to $\mathcal{Q}$. Additionally, the lengths of some of the special edges on P will be doubled; we assume that the algorithm receives a list of all such edges.

For every special edge $e \in E(G)$, the length of e is doubled once e participates in exactly η paths that the algorithm returned, since the last time the length of e was doubled, or since the beginning of the algorithm – whatever happened last.

Lastly, we require that the probability that the algorithm ever returns "FAIL" is bounded by $\frac{1}{2}$. This concludes the definition of the r-restricted s-t-SP problem.

When an algorithm for the problem computes a modified acceptable path P, we may say, for convenience, that it responds to a short-path query. We note that the running time of an algorithm for the s-t-SP problem includes both the time required to maintain its data structures (total update time), and time required to compute and output the modified acceptable paths in every iteration (sometimes referred to as query time).

The following theorem is immediate from the discussion in Section 3.5.1, the definition of the r-restricted s-t-SP problem, and Inequality 1.

Theorem 3.19 *Suppose there is an algorithm for the r-restricted s-t-SP problem, that, given an input $(G, A, B, \Delta, \eta, N)$ with $|V(G)| = n$, has running time at most $n \cdot (n - |B|) \cdot T(n, \Delta, \eta, N)$. Then there is an algorithm for the r-restricted RouteAndCut problem, that, on input $(G, A, B, \Delta, \eta, N)$ with $|V(G)| = n$, has running time at most $O(n \cdot (n - |B|) \cdot T(n, \Delta, \eta, N)) + O(n \cdot (n - |B|))$.*

Lastly, we show a simple but not very efficient algorithm for s-t-SP, that, combined with Theorem 3.19, will be used as the induction base in the proof of Theorem 3.9.

3.5.3 A Simple Algorithm for s-t-SP

In this subsection we provide the following simple algorithm for the r-restricted s-t-SP problem.

Theorem 3.20 *For all $r \geq 1$, there is a deterministic algorithm for the r-restricted s-t-SP problem, that, on input $(G, A, B, \Delta, \eta, N)$ with $|V(G)| = n$, has running time: $O\left(n \cdot (n - |B|) \cdot \frac{(n-|B|)\cdot\eta}{\Delta} \cdot \log^6 N\right)$.*

Proof: We define a dynamic graph H, and we will eventually apply the algorithm for maintaining the ES-Tree data structure from Theorem 2.6 to graph H, with the distance parameter $d = \Lambda$.

We initialize graph H as follows. We let $V(H) = V(G) \cup \{s,t\}$. For every regular edge $e = (u,v) \in E(G)$, we include edge e with length $\tilde{\ell}(e) = 1$ in H. For every special edge $e' = (u',v') \in E(G)$, we include $\lceil \log \Lambda \rceil + 1$ parallel edges $e_1, \ldots, e_{\lceil \log \Lambda \rceil + 1}$, where for all $1 \le i \le \lceil \log \Lambda \rceil + 1$, the length $\tilde{\ell}(e_i) = 2^{i-1}$. Lastly, for every vertex $a \in A$, we add an edge (s,a) of length 1, and for every vertex $b \in B$, we add an edge (b,t) of length 1 to H. Note that graph H can be initialized in time $O(|E(G)| \cdot \log n) \le O(n \cdot (n - |B|) \cdot \log n)$. We then initialize the algorithm for maintaining the ES-Tree data structure from Theorem 2.6 on graph H, with the distance threshold $d = \Lambda$.

Our algorithm consists of at most Δ iterations. In every iteration, we check whether $\mathsf{dist}_H(s,t) \le \Lambda$ holds, in time $O(1)$, using the ES-Tree data structure. Assume first that this is the case. Then we use the ES-Tree data structure to compute the shortest s-t path P in the current graph H, of length at most Λ. From our construction, for every vertex $b \in B$, there is an edge (b,t) of length 1 in H. This ensures that path P may not contain vertices of B as inner vertices. Let P' be the path obtained from P by deleting its first and last vertex. It is then easy to verify that P' is a modified acceptable path, that we return as an output of the current iteration. Let $a \in A$, $b \in B$ be the endpoints of path P'. Recall that following the current iteration, a is deleted from A and b is deleted from B. Additionally, the lengths of some special edges of $E(P')$ may be doubled. We modify graph H, in order to reflect these changes as follows. We delete the edges (s,a) and (b,t) from H. Additionally, for every special edge $e' \in E(P)$ whose length was doubled in G, we delete the shortest-length copy of e' from H.

Assume now that, at the beginning of some iteration, we discover that $\mathsf{dist}_H(s,t) > \Lambda$ holds. Then, in the current graph H, the length of the shortest path connecting a vertex of A to a vertex of B is at least $\Lambda - 2 \ge \frac{\Lambda}{2}$. We now define the lengths $\ell'(e)$ of edges $e \in E(G)$ as follows. If e is a regular edge that lies in H, then we set $\ell'(e) = 0$. If $e = (x,y)$ is a special edge, then we let $\ell'(e)$ be the length of the shortest parallel edge (x,y) in H.

Let A' and B' denote the sets A and B of vertices at the end of the algorithm, respectively. Clearly, $\sum_{e \in E(G)} \ell'(e) \le \sum_{e \in E(G)} \ell(e)$ holds. Next, we claim that, for every path P connecting a vertex $a \in A'$ to a vertex $b \in B'$ in G, the length $\sum_{e \in E(P)} \ell'(e) \ge \frac{\Lambda}{32}$. Indeed, recall that we have established that the length of such a path P in graph H is at least $\frac{\Lambda}{2}$. The only difference between the edge lengths $\left\{\tilde{\ell}(e)\right\}_{e \in E(P)}$ and $\{\ell'(e)\}_{e \in E(P)}$ is that the length $\tilde{\ell}(e)$ of every regular edge $e \in E(H)$ is 1, and $\ell'(e) = 0$. However, since no regular edge may connect a vertex of A to a vertex of B in G, regular and special edges must alternate on path P, and, since the length of every special edge in H is at least 1, the total length $\tilde{\ell}(e)$ of all special edges $e \in E(P)$ is at least $\frac{\sum_{e' \in E(P)} \tilde{\ell}(e')}{4} \ge \frac{\Lambda}{32}$.

Let E^* be the set of all special edges with both endpoints in B'. In order to turn the current assignment $\{\ell'(e)\}_{e \in E(G)}$ of edge lengths to a valid modified cut-witness, we set the length $\ell'(e)$ of every edge $e \in E^*$ to 0. It is easy to verify that, for every path P connecting a vertex of A to a vertex of B in G, its length remains at least $\frac{\Lambda}{32}$, and $\sum_{e \in E(G)} \ell'(e) \le \sum_{e \in E(G) \setminus E^*} \ell(e)$ now holds. Therefore, we obtain a valid modified cut-witness for G.

It now remains to analyze the running time of the algorithm. We start by analyzing the *total update time* – the time required to maintain all data structures, which excludes the *query time* – the time required to compute the modified acceptable paths in every iteration.

Let m denote the total number of edges that are ever present in H, and recall that $m \le O(n \cdot (n - |B|) \cdot \log N)$. Once graph H is initialized, the additional update of the algorithm is bounded by $O(m \cdot \Lambda)$. Therefore, the total update time that is required in order to maintain the data structures (excluding the time required to return the paths in every iteration) is bounded by:

$$O(n \cdot (n-|B|) \cdot \Lambda \cdot \log N) \leq O\left(n \cdot (n-|B|) \cdot \frac{(n-|B|)\cdot \eta}{\Delta} \cdot \log^6 N\right).$$

Assume now that the number of iterations in which the algorithm returned a modified acceptable path is q, and let $P_1, \ldots, P_q$ denote the paths that the algorithm returned. Then the time that the algorithm spent on returning these paths is bounded by $\sum_{i=1}^{q} O(|E(P_i)|)$. Recall that no regular edge connects a vertex of A to a vertex of B in G, and so regular and special edges alternate on each such path P_i. Therefore, for all $1 \leq i \leq q$, $|E(P_i)|$ is asymptotically bounded by the number of special edges on P_i.

Consider now some special edge e of G. When e is first added to G, we include at most $O(\log n)$ copies of e in H. Every time edge e participates in η of the paths that the algorithm returns, one of the copies of e is deleted from H. Therefore, copies of a special edge e of G may participate in at most $O(\eta \log n)$ paths that the algorithm returns, and, overall, $\sum_{i=1}^{q} |E(P_i)| \leq O(n\eta \log n)$ (since there are at most n special edges in G). We conclude that the total running time of the algorithm is bounded by:

$$O\left(n \cdot (n-|B|) \cdot \frac{(n-|B|)\cdot \eta}{\Delta} \cdot \log^6 N\right) + O(n\eta \log N) \leq O\left(n \cdot (n-|B|) \cdot \frac{(n-|B|)\cdot \eta}{\Delta} \cdot \log^6 N\right),$$

since $\eta \leq \Delta \leq n - |B|$ from the problem definition. □

By combining Theorem 3.20 with Theorem 3.19, and recalling that, in the r-restricted RouteAndCut problem, $\frac{(n-|B|)\cdot\eta}{\Delta} \leq 2 \cdot 2^{r \cdot \sqrt{\log N}}$ holds, we obtain the following immediate corollary.

Corollary 3.21 *There is a deterministic algorithm for the r-restricted* RouteAndCut *problem, that, on input $(G, A, B, \Delta, \eta, N)$ with $|V(G)| = n$, has running time at most $O\left(n \cdot (n-|B|) \cdot 2^{r\cdot\sqrt{\log N}} \cdot \log^6 N\right)$.*

We will use this corollary in our inductive proof of Theorem 3.9, for the base case when $r = 1$, and also for general values of r, when the value of the parameter N is sufficiently small.

4 Tools for Expander-Like Graphs and the MaintainCluster Problem

In this section we introduce some tools for expander-like graphs, and define the second main problem that serves as a subroutine in our algorithm for Maximum Bipartite Matching– the MaintainCluster problem. We start by defining subdivided graphs, and providing a new definition of cut sparsity, that extends the standard definition to graphs with edge lengths in a natural way.

4.1 A Subdivided Graph and a New Definition of Cut Sparsity

We first recall a standard definition of cut sparsity in directed graphs. Let G be an arbitrary directed graph, and assume that the graph is unweighted, so all edge lengths and capacities are unit. Under the standard definition, the sparsity of the cut (A, B) in G is defined as $\frac{|E_G(A,B)|}{\min\{|A|,|B|\}}$.

Suppose now that we are given a well-structured graph $G = (L, R, E)$, with a proper **uniform** assignment of lengths to its edges, so the lengths of all special edges in G are unit. In this case, for a well-structured cut (A, B) in G, it would be natural to define its sparsity as $\frac{|E_G(A,B)|}{\min\{|A|,|B|\}}$, to be consistent with the standard definition of sparsity.

Assume now that we are given a well-structured graph $G = (L, R, E)$, with a proper assignment of lengths to its edges, but now the lengths of the special edges may no longer be unit. Several of the expander-based tools that we define below work best when the underlying graph is unweighted; in the context of well-structured graphs, this corresponds to having a proper uniform assignment of edge lengths. A natural way to handle graphs with arbitrary lengths of special edges in this setting is to suitably *subdivide* its edges, so that the lengths of the special edges become unit, but the lengths of all paths are preserved. We now define a subdivided graph of a well-structured graph, a notion that we will use later, that will also motivate our modified definition of cut sparsity.

Definition 4.1 (Subdivided Graph) *Let $G = (L, R, E)$ be a perfect well-structured graph with a proper assignment of lengths $\ell(e)$ to its edges $e \in E$, and let $G' = (L', R', E')$ be another graph. We say that G' is the* subdivided graph *of G, and denote $G' = G^+$, if G' can be obtained from G via the following process. We start with $G' = G$, $L' = L, R' = R$ and $E' = E$, and then consider every special edge $e \in E$ one by one. Let $e = (u, v)$ be any such edge, and denote its length $\ell(e)$ by q. We replace edge e with a path $P(e) = (u = x_1, y_1, x_2, y_2, \ldots, x_q, y_q = v)$. Note that $u \in R'$ and $v \in L'$ must hold. We add vertices $x_2, \ldots, x_q$ to R', and vertices $y_1, \ldots, y_{q-1}$ to L'. Edges $(x_1, y_1), \ldots, (x_q, y_q)$ become special edges, and edges $(y_1, x_2), \ldots, (y_{q-1}, x_q)$ become regular edges of G'. The lengths of the resulting special edges are set to 1, while the lengths of the resulting regular edges are set to 0, so the length of the path $P(e)$ is $q = \ell(e)$.*

Note that, if G is a perfect well-structured graph with proper edge lengths, then its subdivided graph G^+ is a perfect well-structured graph with proper uniform edge lengths. Therefore, we can use the definition of cut sparsity for well-structured graphs with proper uniform edge lengths from above in graph G^+. Every cut (A', B') in graph G^+ naturally also defines a cut (A, B) in G, where we set $A = V(G) \cap A'$ and $B = V(G) \cap B'$. We will define the sparsity of cuts in perfect well-structured graphs with proper edge lengths in a natural way, so that, if a well-structured cut (A', B') in G^+ is φ-sparse according to the standard definition, then the sparsity of the corresponding cut (A, B) in G is at most $O(\varphi)$. Notice that, since graph G^+ is obtained from G by subdividing some of its edges, every edge of G may contribute a number of vertices to A' and/or to B'. We will assign a *weight* to every vertex of G in order to reflect this contribution. We use the following definition of vertex weights throughout the paper.

Definition 4.2 (Vertex Weights) *Let $G = (L, R, E)$ be a perfect well-structured graph with a proper assignment of lengths to its edges. For every vertex $v \in L \cup R$, the* weight *$w(v)$ is defined to be the length of the unique special edge that is incident to v. For a subset S of vertices of G, we define $w(S) = \sum_{v \in S} w(v)$, and for a subgraph G' of G we define $w(G') = w(V(G'))$.*

We are now ready to define the sparsity of cuts in perfect well-structured graphs.

Definition 4.3 (Cut Sparsity) *Let $G = (L, R, E)$ be a perfect well-structured graph with a proper assignment of lengths on its edges, and let (A, B) be a well-structured cut in G. The* sparsity *of cut (A, B) is:*

$$\Phi_G(A, B) = \frac{|E_G(A, B)|}{\min\{w(A), w(B)\}}.$$

We will use the following simple observation whose proof can be found in Section A.1 of the Appendix.

Observation 4.4 *Let $G = (L, R, E)$ be a perfect well-structured graph with proper lengths $\ell(e)$ on its edges $e \in E$, and let G^+ be its subdivided graph. Assume further that we are given a parameter*

$0 < \varphi < 1$, and that the length of every edge in G is at most $\frac{1}{4\varphi}$. Let (A', B') be a well-structured cut in G^+ of sparsity at most φ, and let (A, B) be the corresponding cut in G, where $A = A' \cap V(G)$ and $B = B' \cap V(G)$. Then (A, B) is a well-structured cut in G, $w(A) \geq |A'|/8$, $w(B) \geq |B'|/8$, and $\Phi_G(A, B) \leq 8\varphi$.

The next observation allows us to transform a weakly well-structured cut into a strongly well-structured cut in a perfect well-structured graph, while roughly preserving its sparsity. The proof appears in Section A.2 of the Appendix.

Observation 4.5 *Let $G = (L, R, E)$ be a perfect well-structured graph with a proper assignment of lengths $\ell(e)$ to its edges $e \in E$, and let $0 < \varphi \leq 1$ be a parameter. Let $W = \sum_{e\in E} \ell(e)$, and assume that, for all $e \in E$, $\ell(e) \leq \frac{W}{2}$. Let (A, B) be a $\frac{\varphi}{2}$-sparse weakly well-structured cut in G. Then there is a φ-sparse strongly well-structured cut (A', B') in G with $w(A') \geq w(A)/4$ and $w(B') \geq w(B)/4$. Moreover, there is a deterministic algorithm, that, given G, φ, and cut (A, B) in G, computes cut (A', B') in G with the above properties in time $O(\min\{\mathrm{Vol}_G(A), \mathrm{Vol}_G(B)\})$.*

4.2 The MaintainCluster Problem

We abstract the central technical tool that we use in order to maintain expander-like graphs by the MaintainCluster problem, that we define next.

As an input to the MaintainCluster problem, we are given a **perfect** well-structured graph G with a proper assignment of lengths $\ell(e)$ to its edges $e \in E(G)$, that subsequently undergoes some updates, though G remains a perfect well-structured graph throughout the algorithm. Recall that we have defined vertex weights for perfect well-structured graphs, where the weight of a vertex v is the length of the unique special edge incident to v. For every vertex $v \in V(G)$, we denote by $w^0(v)$ its initial weight in the initial input graph G, and we denote $W^0(G) = \sum_{v\in V(G)} w^0(v)$. We are now ready to define the MaintainCluster problem.

Definition 4.6 *The input to the* MaintainCluster *problem consists of a perfect well-structured graph $G = (L, R, E)$ with $|V(G)| = n$, together with a proper assignment of lengths $\ell(e)$ to its edges $e \in E$, and parameters $N \geq W^0(G)$, $1 \leq \Delta \leq N$, $1 \leq \eta \leq \Delta$, and $d \geq (\log N)^{64}$, such that $\frac{\Delta\cdot d}{\eta} \leq n$, and N is greater than a sufficiently large constant, so that, for example, $N > 2^{2^{c_2}}$ holds, where c_2 is the constant from Theorem 3.9.*

The algorithm consists of at most Δ iterations. At the beginning of every iteration i, the algorithm is given two vertices $x, y \in V(G)$, and it must return a simple path P_i in G connecting x to y of length at most d. (We may sometimes say that the algorithm is given a short-path *query between the pair (x, y) of vertices, and it returns the path P_i in response to the query). After that the length of some special edges on path P_i may be doubled. We are guaranteed that the following property holds:*

P1. If $\tau' > \tau$ are two times at which the length of some edge $e \in E(G)$ is doubled, then the algorithm returned at least η paths in response to queries during the time interval $(\tau, \tau']$ that contained e.

The algorithm may, at any time, produce a strongly well-structured cut (X, Y) in G of sparsity $\Phi_G(X, Y) \leq \frac{(\log N)^{64}}{d}$. If $w(X) \leq w(Y)$, then let $Z = X$ and $Z' = Y$; otherwise, let $Z = Y$ and $Z' = X$. We also let $J' \subseteq Z'$ denote the set of vertices that serve as endpoints of the edges of $E_G(X, Y)$. The vertices of $Z \cup J'$ are then deleted from G, and the algorithm continues with the resulting graph $G = G[Z' \setminus J']$. Note that we are guaranteed that G remains a perfect well-structured *graph after this modification. Once $|V(G)| \le n/2$ holds, the algorithm terminates, even if fewer than Δ iterations have passed.*

The algorithm may also, at any time, return "FAIL". The probability that the algorithm ever returns "FAIL" must be bounded by $1/N^4$.

For an integer $1 \le r \le \lceil\sqrt{\log N}\rceil$, we say that an instance (G, Δ, η, d, N) of the MaintainCluster *problem is r-restricted, if $d \le 2^{r\cdot\sqrt{\log N}}$.*

We prove the following theorem that provides an algorithm for the MaintainCluster problem. Though we do not use the theorem directly, it easily follows from our results and we believe that it is interesting in its own right.

Theorem 4.7 *There is a randomized algorithm for the* MaintainCluster *problem, that, on an instance (G, Δ, η, d, N) of the problem, has running time at most $O\left(W^0(G)\cdot(W^0(G)+\Delta)\cdot 2^{O(\sqrt{\log N}\cdot\log\log N)}\right)$.*

4.3 The MaintainSpecialCluster Problem

We will sometimes need to use a slight variation of the MaintainCluster problem, that we refer to as the MaintainSpecialCluster problem. In this problem, as before, we are given a perfect well-structured graph G with proper lengths $\ell(e)$ on its edges $e \in E(G)$, that subsequently undergoes some updates, though G remains a perfect well-structured graph throughout the algorithm. Graph G is given in the adjacency-list representation, and initially all special edges have length 1. For every vertex $v \in V(G)$, we define its initial weight $w^0(v)$ exactly as before, and we denote $W^0(G) = \sum_{v\in V(G)} w^0(v)$ as before. In addition to the perfect well-structured graph G, the algorithm is given as input a subset β of vertices of G. Over the course of the algorithm, some vertices may be deleted from the set β, but we will ensure that, throughout the algorithm, every special edge incident to a vertex in β has length 1. For intuition, the vast majority of the vertices of the input graph G lie in β, and this property is preserved throughout the entire algorithm. Unlike the MaintainCluster problem, in a short-path query, we are now given a single vertex $x \in V(G)$, and we need to return a path P in the current graph G that connects x to any vertex of β, such that no inner vertices of P lie in β. The last endpoint of the path P is then deleted from β, and the lengths of some edges of the path P may be doubled, like in the MaintainCluster problem. Like in the MaintainCluster problem, the algorithm may, at any time, return a sparse strongly well-structured cut (X, Y) in G, but now we require that $w(X) \ge 1.8w(\beta \cap X) = 1.8|\beta \cap X|$, that is, much of the weight of the vertices in X must come from vertices that do not lie in β. The vertices of X, and all endpoints of the edges in $E_G(X, Y)$ are then deleted from G, and the algorithm continues. We are now ready to define the MaintainSpecialCluster problem formally.

Definition 4.8 *The input to the* MaintainSpecialCluster *problem consists of a perfect well-structured graph $G = (L, R, E)$ with a proper assignment of lengths $\ell(e)$ to its edges $e \in E$, that are given in the adjacency-list representation, such that the length of every special edge is 1. Additionally, we are given a subset $\beta \subseteq V(G)$ of vertices with $|\beta| \ge \frac{99|V(G)|}{100}$, and parameters $N \ge W^0(G)$, $1 \le \Delta \le \frac{|\beta|}{(\log N)^{128}}$, $1 \le \eta \le \Delta$, and $d \ge (\log N)^{64}$, such that $\frac{\Delta\cdot d}{\eta} \le |V(G)| - |\beta| + \Delta\cdot(\log N)^{64}$, and N is greater than a sufficiently large constant. Over the course of the algorithm, vertices may be deleted from set β, and the lengths of some special edges may be increased, but the length of every special edge incident to a vertex that currently lies in β is always 1.*

The algorithm consists of at most Δ iterations. At the beginning of every iteration i, the algorithm is given a vertex $x \in V(G)$, and it must return a simple path P_i in G connecting x to any vertex $y \in \beta$, such that the length of the path is at most d, and no inner vertices on the path lie in β (if $x \in \beta$, that *we require that path P_i only consists of the vertex x). After that, the length of some special edges on path P_i may be doubled, and vertex y is deleted from β. We are guaranteed that Property P1 from the definition of the* MaintainCluster *problem holds.*

The algorithm may, at any time, produce a strongly well-structured cut (X, Y) in G of sparsity $\Phi_G(X, Y) \leq \frac{(\log N)^{64}}{d}$, such that $w(X) \geq 1.8|\beta \cap X|$ holds. Let $J' \subseteq Y$ denote the set of vertices that serve as endpoints of the edges of $E_G(X, Y)$. The vertices of $X \cup J'$ are then deleted from G, and the algorithm continues with the resulting graph $G = G[Y \setminus J']$. Note that we are guaranteed that G remains a perfect well-structured graph after this modification.

The algorithm may also, at any time, return "FAIL". The probability that the algorithm ever returns "FAIL" must be bounded by $1/4$. Lastly, if the number of vertices in β decreases by a factor 2, the algorithm terminates.

For an integer $1 \leq r \leq \lceil\sqrt{\log N}\rceil$, we say that an instance $(G, \beta, \Delta, \eta, d, N)$ of the MaintainSpecialCluster *problem is r-restricted, if $d \leq 2^{r \cdot \sqrt{\log N}}$.*

We prove the following theorem that provides an algorithm for the MaintainSpecialCluster problem.

Theorem 4.9 *There is a randomized algorithm for the* MaintainSpecialCluster *problem, that, on an instance $(G, \beta, \Delta, \eta, d, N)$ of the problem, has running time at most:*

$$O\left(W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{O(\sqrt{\log N} \cdot \log\log N)}\right).$$

In order to prove Theorems 4.7 and 4.9, we use induction. Specifically, we will consider r-restricted instances of both problems problem, from smaller to larger values of r. The following theorem summarizes our main result for the r-restricted MaintainCluster and MaintainSpecialCluster problems. It uses the large enough constants $c_1 \gg c_2$ that also appear in the statment of Theorem 3.9.

Theorem 4.10 *For all $r \geq 1$, there is a randomized algorithm for the r-restricted* MaintainCluster *problem, that, on an instance (G, Δ, η, d, N) of the problem, has running time at most:*

$$c_1 \cdot (W^0(G))^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r} + c_1 W^0(G) \cdot \Delta \cdot \log^4 N,$$

and there is a randomized algorithm for the the r-restricted MaintainSpecialCluster *problem, that, on an instance $(G, \beta, \Delta, \eta, d, N)$ of the problem, has running time at most:*

$$c_1 \cdot W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r}.$$

Note that the proof of Theorem 4.7 easily follows from Theorem 4.10. Indeed, consider any instance (G, Δ, η, d, N) of the MaintainCluster problem. From the definition of the problem, $d \leq \frac{n\eta}{\Delta} \leq n \leq N$ must hold, and so in particular $d \leq 2^{r \cdot \sqrt{\log N}}$, for $r = \lceil\sqrt{\log N}\rceil$. Therefore, any instance of the problem is r-restricted for $r = \lceil\sqrt{\log N}\rceil$. By applying Theorem 4.10 with $r = \lceil\sqrt{\log N}\rceil$, we obtain an algorithm for the given instance of the MaintainCluster problem, whose running time is bounded by:

$$\begin{aligned} O&\left(W^0(G) \cdot (W^0(G) + \Delta) \cdot 2^{O(\sqrt{\log N})} \cdot (\log N)^{O(\sqrt{\log N})}\right) \\ &\leq O\left(W^0(G) \cdot (W^0(G) + \Delta) \cdot 2^{O(\sqrt{\log N} \cdot \log\log N)}\right), \end{aligned}$$

as required.

Similarly, the proof of Theorem 4.9 easily follows from Theorem 4.10. Indeed, consider any instance $(G, \beta, \Delta, \eta, d, N)$ of the MaintainSpecialCluster problem, and denote $n = |V(G)|$. As before, from the definition of the problem, $d \leq \frac{n\eta}{\Delta} \leq n \leq N$ must hold, and so in particular $d \leq 2^{r \cdot \sqrt{\log N}}$, for $r = \lceil \sqrt{\log N} \rceil$. Therefore, any instance of the problem is r-restricted for $r = \lceil \sqrt{\log N} \rceil$. By applying Theorem 4.10 with $r = \lceil \sqrt{\log N} \rceil$, we obtain an algorithm for the given instance of the MaintainSpecialCluster problem, whose running time is bounded by:

$$
\begin{aligned}
&O\left(W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{O(\sqrt{\log N})} \cdot (\log N)^{O(\sqrt{\log N})}\right) \\
&\qquad \leq O\left(W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{O(\sqrt{\log N} \cdot \log\log N)}\right).
\end{aligned}
$$

It now remains to complete the proofs of Theorems 3.9 and 4.10.

4.4 Completing the Proofs of Theorem 3.9 and Theorem 4.10

We prove the two theorems together, by induction on r. Specifically, we first show that, given an efficient algorithms for the r-restricted MaintainCluster and MaintainSpecialCluster problems, we can obtain an efficient algorithm for the $(r+1)$-restricted RouteAndCut problem. This result is summarized in the following theorem, whose proof is deferred to Section 5.

Theorem 4.11 *Suppose that, for some parameter $r \geq 1$, there exists a randomized algorithm for the r-restricted* MaintainCluster *problem, that, given as input an instance (G, Δ, η, d, N) of the problem, has running time at most: $c_1 \cdot (W^0(G))^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r} + c_1 \cdot W^0(G) \cdot \Delta \cdot \log^4 N$. Assume further that there also exists a randomized algorithm for the r-restricted* MaintainSpecialCluster *problem, that, on an instance $(G, \beta, \Delta, \eta, d, N)$ of the problem, has running time at most: $c_1 \cdot W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r}$. Then there is a randomized algorithm for the $(r+1)$-restricted* RouteAndCut *problem, that, given as input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most $c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r + 8c_2}$.*

Next, we show that an efficient algorithm for the r-restricted RouteAndCut problem implies an efficient algorithm for the r-restricted MaintainCluster and MaintainSpecialCluster problems, in the following theorem, whose proof is deferred to Sections 6 and 7.

Theorem 4.12 *Suppose that, for some parameter $r \geq 1$, there exists a randomized algorithm for the r-restricted* RouteAndCut *problem, that, given as an input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most $c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2}$. Then there exists a randomized algorithm for the r-restricted* MaintainCluster *problem, that, on input (G, Δ, η, d, N) has running time at most: $c_1 \cdot (W^0(G))^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r} + c_1 \cdot W^0(G) \cdot \Delta \cdot \log^4 N$, and there exists a randomized algorithm for the r-restricted* MaintainSpecialCluster *problem, that, on input $(G, \beta, \Delta, \eta, d, N)$ has running time at most: $c_1 \cdot W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r}$.*

The majority of the remainder of this paper is dedicated to the proofs of Theorem 4.11 and Theorem 4.12. The proofs of Theorem 3.9 and Theorem 4.10 easily follow from these theorems, as we show next.

Recall that, from Corollary 3.21, there is an algorithm for the 1-restricted RouteAndCut problem, that, given as input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most

$O\left(n \cdot (n-|B|) \cdot 2^{r \cdot \sqrt{\log N}} \cdot \log^6 N\right) \leq c_1 \cdot n \cdot (n-|B|) \cdot 2^{\sqrt{\log N}} \cdot (\log N)^{8c_2}$, since we can assume that c_1 is a large enough constant. This proves the induction base for the RouteAndCut problem. From Theorem 4.12 we then conclude that there exists a randomized algorithm for the 1-restricted MaintainCluster problem, that, on input (G, Δ, η, d, N) has running time at most: $c_1 \cdot (W^0(G))^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2} + c_1 \cdot W^0(G) \cdot \Delta \cdot \log^4 N$, and that there is the 1-restricted MaintainSpecialCluster problem, that, on input $(G, \beta, \Delta, \eta, d, N)$ has running time at most: $c_1 \cdot W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2}$. This proves the induction base for the MaintainCluster and the MaintainSpecialCluster problems.

For the induction step, consider an integer $r \geq 1$, and assume that both Theorem 3.9 and Theorem 4.10 hold for r-restricted instances. Then from Theorem 4.11, there is a randomized algorithm for the $(r+1)$-restricted RouteAndCut problem, that, given as input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most $c_1 \cdot n \cdot (n-|B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r + 8c_2}$, proving Theorem 3.9 for $(r+1)$-restricted instances. By applying Theorem 4.12 with integer $(r+1)$, we then conclude that there exists a randomized algorithm for the $(r+1)$-restricted MaintainCluster problem, that, on input (G, Δ, η, d, N) has running time at most: $c_1 \cdot (W^0(G))^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r+1)} + c_1 \cdot W^0(G) \cdot \Delta \cdot \log^4 N$, and that there exists a randomized algorithm for the $(r+1)$-restricted MaintainSpecialCluster problem, that, on input $(G, \beta, \Delta, \eta, d, N)$ has running time at most: $c_1 \cdot W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r+1)}$. This proves proving Theorem 4.10 for $(r+1)$-restricted instances of MaintainCluster and MaintainSpecialCluster. This completes the proofs of Theorems 3.9 and 4.10 from Theorems 4.11 and 4.12. In the next three sections we prove Theorem 4.11 and Theorem 4.12.

5 From r-Restricted MaintainCluster and MaintainSpecialCluster to $(r+1)$-Restricted RouteAndCut: Proof of Theorem 4.11

The proof of Theorem 4.11 follows from the following theorem:

Theorem 5.1 *Suppose that, for some parameter $r \geq 1$, there exists a randomized algorithm for the r-restricted* MaintainCluster *problem, that, given as input an instance (G, Δ, η, d, N) of the problem, has running time at most: $c_1 \cdot (W^0(G))^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r} + c_1 \cdot W^0 \cdot \Delta \cdot \log^4 N$. Suppose further that there exists a randomized algorithm for the r-restricted* MaintainSpecialCluster *problem, that, given as input an instance $(G, \beta, \Delta, \eta, d, N)$ of the problem, has running time at most: $c_1 \cdot W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r}$. Then there is a randomized algorithm for the $(r+1)$-restricted s-t-*SP *problem, that, given an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running running time at most $c_1 \cdot n \cdot (n-|B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r + 7c_2}$.*

Before we provide the proof of Theorem 5.1, we show that the proof of Theorem 4.11 follows from it.

Proof of Theorem 4.11: Consider a parameter $r \geq 1$, and assume that there exists a randomized algorithm for the r-restricted MaintainCluster problem, that, on input (G, Δ, η, d, N), has running time at most: $c_1 \cdot (W^0(G))^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r} + c_1 \cdot W^0 \cdot \Delta \cdot \log^4 N$. Assume further that there exists a randomized algorithm for the r-restricted MaintainSpecialCluster problem, that, on input $(G, \beta, \Delta, \eta, d, N)$, has running time at most: $c_1 \cdot W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r}$. From Theorem 5.1, there exists a randomized algorithm for the $(r+1)$-restricted s-t-SP problem, that, on input $(G, A, B, \Delta, \eta, N)$ with $|V(G)| = n$, has running running time at most $c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r + 7c_2}$. From Theorem 3.19, there exists a randomized algorithm for the $(r+1)$-restricted RouteAndCut problem, that, on input (G, A, B, Δ, η) with $|V(G)| = n$, has running time is at most:

$$O\left(c_1\cdot n\cdot(n-|B|)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+7c_2}\right)+O(n\cdot(n-|B|))$$
$$\leq c_1\cdot n\cdot(n-|B|)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+8c_2},$$

since c_2 is a sufficiently large constant. □

From now on we focus on the proof of Theorem 5.1. We denote the randomized algorithms for the r-restricted MaintainCluster and MaintainSpecialCluster problems from the theorem statement by $\mathcal{A}$ and $\mathcal{A}'$, respectively.

We denote the input to the $(r+1)$-restricted s-t-SP problem by $(G, A, B, \Delta, \eta, N)$, where $G=(L,R,E)$ is an n-vertex well-structured graph, A and B are two disjoint subsets of vertices of G, and $N\geq n$, $\Delta\geq 1$, and $1\leq\eta\leq\Delta$ are parameters. Recall that no regular edge of G connects a vertex of A to a vertex of B, $\Delta\leq n-|B|$, $|E|\leq O(n\cdot(n-|B|))$, and $\frac{(n-|B|)\cdot\eta}{\Delta}\leq 2\cdot 2^{(r+1)\sqrt{\log N}}$ hold.

Recall that, as the algorithm progresses, we maintain lengths $\ell(e)$ for edges $e\in E(G)$, and, additionally, some vertices may leave the sets A and B. Since the number of iterations is bounded by Δ, at most Δ vertices may be deleted from A and from B. For convenience, we denote by A^0 and B^0 the initial sets A and B, respectively, to distinguish them from the sets that are maintained during the algorithm.

Note that we can assume that N is greater than a sufficiently large constant, so that, for example, $N>2^{2^{c_2}}$ holds. Indeed, if this is not the case, then, from Theorem 3.20, we obtain a determinisic algorithm that solves the input instance of s-t-SP in time:

$$O\left(n\cdot(n-|B^0|)\cdot\frac{(n-|B^0|)\cdot\eta}{\Delta}\cdot\log^6 N\right)\leq O\left(n\cdot(n-|B^0|)\cdot N\cdot\log^6 N\right)$$
$$\leq c_1\cdot n\cdot(n-|B^0|)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+7c_2},$$

since $n\leq N\ll c_1$, and $\eta\leq\Delta$ (recall that, by definition, $c_1\gg c_2$ holds). Therefore, we assume from now on that N is sufficiently large.

Additionally, we can assume that $\frac{(n-|B^0|)\cdot\eta}{\Delta}>2^{4\sqrt{\log N}}$, since otherwise, from Theorem 3.20, we obtain a determinisic algorithm that solves the input instance of s-t-SP in time:

$$O\left(n\cdot(n-|B^0|)\cdot\frac{(n-|B^0|)\cdot\eta}{\Delta}\cdot\log^6 N\right)\leq O\left(n\cdot(n-|B^0|)\cdot 2^{4\sqrt{\log N}}\cdot\log^6 N\right)$$
$$\leq c_1\cdot n\cdot(n-|B^0|)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+7c_2}.$$

Therefore, we assume from now on that:

$$\frac{(n-|B^0|)\cdot\eta}{\Delta}>2^{4\sqrt{\log N}} \tag{2}$$

Throughout the algorithm, we use the parameter $\Lambda=(n-|B^0|)\cdot\frac{\eta\cdot\log^5 n}{\Delta}$. We let H be the graph that is obtained from G, by adding a source vertex s, that connects to every vertex $a\in A$ with a length-0 edge, and a destination vertex t, to which every vertex $b\in B$ connects with a length-0 edge. If an edge $e\in E(G)$ is a special edge for graph G, then e remains a special edge for H. All other edges of H, including edges incident to s and t, are called regular edges. For every edge $e\in E(G)$, its length in H remains the same as in G, so for convenience, we denote the length of every edge $e \in E(H)$ by $\ell(e)$.

At the beginning of the algorithm, for each edge $e \in E(H)$, its length $\ell(e)$ is set as follows: if e is a regular edge, then $\ell(e) = 0$, and if e is a special edge, then $\ell(e) = 1$. The algorithm proceeds as follows. We start by setting $\mathcal{Q} = \emptyset$, and then perform at most Δ iterations. In every iteration, the algorithm must do one of the following:

R1. either return FAIL;

R2. or return a simple s-t path P of length at most Λ in the current graph H, such that P contains exactly one vertex of B – the penultimate vertex on the path.

R3. or compute an assignment $\ell'(e) \geq 0$ of lengths to all edges $e \in E(H)$, such that, if we denote by $\tilde{E}$ the set of all special edges of G with both endpoints in B, then $\sum_{e \in E(H)} \ell'(e) \leq \frac{\Lambda \cdot \Delta}{\eta \log^4 n} + \sum_{e \in E(H) \setminus \tilde{E}} \ell(e)$; the length $\ell'(e)$ of every edge e that is incident to s or to t is 0; and the distance from s to t in H, with respect to edge lengths $\ell'(\cdot)$, is at least $\frac{\Lambda}{32}$;

Note that, if the algorithm returns assignments of lengths $\ell'(e)$ to edges $e \in E(H)$ as in Outcome R3, then these assignments of edge lengths define a modified cut-witness for G (see Definition 3.18). In this case, the algorithm terminates, and returns the lengths $\ell'(e)$ of all edges $e \in E(H \setminus \{s,t\})$. Slightly abusing the notation, for the sake of simplicity, we will call an assignment of lengths $\ell'(e)$ to the edges $e \in E(H)$ as described in Outcome R3 a *cut-witness*.

If the algorithm returns an s-t path P as in Outcome R2, then $P \setminus \{s,t\}$ is a modified acceptable path for G (see Definition 3.17). To simplify the notation, we say that a path P described in Outcome R2 is an *acceptable path*. If the algorithm returns an acceptable path P, then we let $P' = P \setminus \{s,t\}$, and we let $a \in A$, $b \in B$ be the endpoints of path P'. We remove a from A and b from B, add P' to $\mathcal{Q}$, and delete the edges (s,a) and (b,t) from H. Additionally, the lengths of some of the special edges on P may be doubled; we assume that the algorithm receives the list of all such edges.

From the problem definition, for every special edge $e \in E(H)$, the length of e is doubled once e participates in exactly η paths that the algorithm returned since the last time the length of e was doubled, or since the beginning of the algorithm – whatever happened last.

Lastly, in the case of Outcome R1, we terminate the algorithm and return "FAIL", but we require that the probability that the algorithm ever returns "FAIL" is bounded by $\frac{1}{2}$.

Some Useful Bounds

We denote by E^{spec} the set of all special edges in graph G, and we denote by $E^* \subseteq E^{\mathsf{spec}}$ the set of all special edges $e = (u,v)$, such that $u, v \in B$ holds at the end of the algorithm. Recall that the length of a special edge e may only be increased once it participates in some path that the algorithm returns, and, if a vertex $x \in B$ lies on any path that the algorithm returns, then x is deleted from B. Therefore, every edge $e \in E^*$ has length 1 throughout the algorithm. We start by bounding $|E^{\mathsf{spec}} \setminus E^*|$ in the following simple observation.

Observation 5.2 $|E^{\mathsf{spec}} \setminus E^*| \leq 2(n - |B^0|)$.

Proof: We partition the set $E^{\mathsf{spec}} \setminus E^*$ if edges into two subsets: set E_1 contains all edges e, such that at least one endpoint of e lies in $V(G) \setminus B^0$, and set E_2 contains all remaining edges of $E^{\mathsf{spec}} \setminus E^*$.

Since every edge of E_1 is incident to a vertex of $V(G) \setminus B^0$, and every vertex of G may serve as an endpoint of at most one special edge, $|E_1| \le n - |B^0|$ must hold. Consider now some edge $e \in E_2$. From the definition, over the course of the algorithm, at least one endpoint of e was deleted from B. Since at most Δ vertices are deleted from B over the course of the algorithm, and since each such vertex may serve as an endpoint of at most one special edge, we get that $|E_2| \le \Delta \le n - |B^0|$, from the definition of the s-t-SP problem. Altogether, we get that $|E^{\mathsf{spec}} \setminus E^*| \le |E_1| + |E_2| \le 2(n - |B^0|)$. $\square$

For each edge $e \in E^{\mathsf{spec}}$, let $\ell^*(e)$ be the length of e at the end of the algorithm. In the next simple observation we bound $\sum_{e \in E^{\mathsf{spec}}} \ell^*(e)$.

Observation 5.3 *$\sum_{e \in E^{\mathsf{spec}}} \ell^*(e) \le |E^{\mathsf{spec}}| + 2(n - |B^0|) \log^5 n$, and $\sum_{e \in E^{\mathsf{spec}} \setminus E^*} \ell^*(e) \le 4(n - |B^0|) \log^5 n$.*

Proof: Recall that, at the beginning of the algorithm, for every edge $e \in E^{\mathsf{spec}}$, we set $\ell(e) = 1$. Let $E' \subseteq E^{\mathsf{spec}}$ be the set of all edges $e \in E^{\mathsf{spec}}$ with $\ell^*(e) > 1$. Clearly, $\sum_{e \in E^{\mathsf{spec}}} \ell^*(e) \le |E^{\mathsf{spec}}| + \sum_{e \in E'} \ell^*(e)$.

Let q be the total number of iterations of the algorithm, so that, at the end of the algorithm, $|\mathcal{Q}| = q$ holds, and let $\mathcal{Q} = \{P_1, \ldots, P_q\}$, where the paths are indexed in the order in which they were added to $\mathcal{Q}$. For all $1 \le i \le q$, let C_i denote the length of path P_i when it was added to $\mathcal{Q}$. Clearly, $C_i \le \Lambda$, and $\sum_{i=1}^q C_i \le q \cdot \Lambda \le \Delta \cdot \Lambda$.

Consider now some edge $e \in E'$, and recall that $\ell(e)$ was doubled at least once during the algorithm. Let τ be the last time when the length of e was doubled. Then, prior to time τ, there were at least η iterations i, during which the path P_i that was added to $\mathcal{Q}$ contained e, and the length of e during the corresponding iteration was $\frac{\ell^*(e)}{2}$. In other words, edge e contributes at least $\frac{\ell^*(e)\eta}{2}$ to $\sum_{i=1}^q C_i$. Altogether, we get that $\sum_{i=1}^q C_i \ge \sum_{e \in E'} \frac{\ell^*(e)\eta}{2}$. Therefore:

$$\sum_{e \in E'} \ell^*(e) \le \frac{2\sum_{i=1}^q C_i}{\eta} \le \frac{2\Delta\Lambda}{\eta} \le 2(n - |B^0|) \cdot \log^5 n,$$

since $\Lambda = (n - |B^0|) \cdot \frac{\eta \cdot \log^5 n}{\Delta}$. Altogether, we get that:

$$\sum_{e \in E^{\mathsf{spec}}} \ell^*(e) \le |E^{\mathsf{spec}}| + \sum_{e \in E'} \ell^*(e) \le |E^{\mathsf{spec}}| + 2(n - |B^0|) \log^5 n.$$

Lastly, from Observation 5.2, $|E^{\mathsf{spec}}| - |E^*| \le 2(n - |B^0|)$, and so:

$$\sum_{e \in E^{\mathsf{spec}} \setminus E^*} \ell^*(e) \le \sum_{e \in E^{\mathsf{spec}}} \ell^*(e) - |E^*| \le |E^{\mathsf{spec}}| + 2(n - |B^0|) \log^5 n - |E^*| \le 4(n - |B^0|) \log^5 n.$$

$\square$

Throughout the algorithm, we use a parameter $N' = \max\left\{N, 8n \cdot \log^5 n\right\}$. Let τ be any time in the algorithm's execution, and let H' be any subgraph of $H \setminus \{s, t\}$, that is a perfect well-structured graph. Recall that the weight of every vertex $v \in V(H')$ is the length of the unique special edge incident to v. From Observation 5.3, and since $|E^{\mathsf{spec}}| \le n$, the total weight of all vertices of H' must be bounded by N'. Notice also that, since $N \ge n$, we get that $N' \le 8N \log^5 n \le N \log^6 N$. It is then easy to see that:

$$\log N' \leq \log N + 6\log\log N \leq \left(1 + \frac{1}{64c_2 r}\right)\log N, \tag{3}$$

since $r \leq \left\lceil \sqrt{\log N} \right\rceil$ and N is large enough. Additionally, $\sqrt{\log N'} \leq \sqrt{\log N + 6\log\log N} \leq \sqrt{\log N} + \sqrt{6\log\log N}$. Therefore:

$$2^{\sqrt{\log N'}} \leq 2^{\sqrt{\log N}} \cdot \log^6 N \leq 2^{2\sqrt{\log N}}. \tag{4}$$

We will use these facts later.

Algorithm for the $(r+1)$-restricted s-t-SP problem: The description of our algorithm for the $(r+1)$-restricted s-t-SP problem consists of two parts. In the first part, we provide an algorithm for maintaining an $\mathcal{ATO}$ for graph H. This algorithm in turn exploits algorithm $\mathcal{A}$ for the r-restricted MaintainCluster problem and Algorithm $\mathcal{A}'$ for the r-restricted MaintainSpecialCluster problem. In the second part, we use the algorithm from Theorem 2.7 for the DAG-like SSSP problem on the corresponding contracted graph.

5.1 Part 1 - Maintaining the $\mathcal{ATO}$

In this part we describe our algorithm for maintaining the $\mathcal{ATO}$ $(\mathcal{X}, \rho, U, \gamma)$ for graph H. Notice that $|V(H)| = |V(G)| + 2 = n + 2$. Throughout, we use a parameter $d = \frac{(n-|B^0|)\cdot\eta}{4\Delta\cdot 2^{\sqrt{\log N}}}$. Notice that:

$$2^{2\sqrt{\log N}} \leq d \leq 2^{r\sqrt{\log N}-1}. \tag{5}$$

Here, the first inequality follows from Inequality 2. The second inequality follows since the input instance of s-t-SP is $(r+1)$-restricted, so $\frac{(n-|B^0|)\cdot\eta}{\Delta} \leq 2 \cdot 2^{(r+1)\sqrt{\log N}}$ holds.

Let $U \subseteq V(H) \setminus \{s,t\}$ be the collection of all vertices v, such that some special edge of E^{spec} is incident to v, and let $U' = V(H) \setminus U$. Notice that $H[U]$ is a perfect well-structured graph (see Definition 3.1). We let the initial collection $\mathcal{X}$ of vertex subsets contain the set U, and, for every vertex $v \in U'$, a set $X_v = \{v\}$. For convenience, we denote $S = \{s\}$ and $T = \{t\}$. We distinguish between two cases. Case 1 happens when $|U| > 2(n - |B^0|)\cdot(\log N)^{128}$. In this case, we let U be the special cluster of the $\mathcal{ATO}$, and we set $\gamma = |U| - 256(n-|B^0|)(\log N)^{64}$. Otherwise, Case 2 happens, the special cluster of the $\mathcal{ATO}$ is undefined, and we set $\gamma = 1$.

Next, we assign, to every resulting set $X \in \mathcal{X}$, an interval $I_X \subseteq \{1, \ldots, |V(H)| - \gamma + 1\}$, so that, for each regular cluster $X \in \mathcal{X}$, $|I_X| = |X|$, and, if the special cluster is defined, then $|I_U| = |U| - \gamma + 1$. In order to do so, it is sufficient to define an ordering σ of the sets in $\mathcal{X}$. We let S appear first in the ordering, followed by the sets in $\{X_v \mid v \in U' \cap L\}$ in an arbitrary order, followed by set U, which is followed by the sets in $\{X_v \mid v \in U' \cap R\}$ in an arbitrary order, followed by T. Since vertices of U' are not incident to special edges, it is easy to verify that no edge of H is a right-to-left edge with respect to this initial $\mathcal{ATO}$. Throughout the algorithm, we refer to each set $X \in \mathcal{X}$ of vertices as a *cluster*. If $|X| = 1$, then we say that X is a *singleton* cluster. As the algorithm progresses, cluster U may be partitioned into smaller and smaller clusters, with each such cluster assigned an interval that is contained in the initial interval I_U, but the clusters in $\{X_v \mid v \in U'\}$, together with their corresponding intervals I_{X_v}, remain unchanged. For convenience, we denote the initial set U by U^0. We will ensure that, throughout the algorithm, the following invariants hold:

I1. If $X \in \mathcal{X}$ is a non-singleton cluster, then $H[X]$ is a perfect well-structured graph;

I2. If the special cluster is defined, then $U \in \mathcal{X}$ holds throughout the algorithm, and moreover, $|U| \geq \gamma$ always holds; and

I3. If $e = (x, y)$ is a right-to-left edge with respect to the current $\mathcal{ATO}$, and $X, Y \in \mathcal{X}$ are the clusters that contain x and y, respectively, so that Y appears before X in the ordering σ of clusters of $\mathcal{X}$ defined by the $\mathcal{ATO}$, then e must be a special edge, and moreover, $X = \{x\}$ and $Y = \{y\}$ must hold; in other words, X and Y are both singleton clusters.

It is easy to verify that the invariant holds for the initial $\mathcal{ATO}$.

For each cluster X that is ever added to $\mathcal{X}$, we denote by X^0 the set X when it was just added to $\mathcal{X}$. If $|X^0| > 1$ (that is, X is not a singleton cluster when it is added to $\mathcal{X}$), then, from Invariant I1, $H[X]$ is a perfect well-structured graph, so weights of vertices of X are well defined. In such a case, we denote by $W^0(X)$ the total weight of all vertices of X when X is added to $\mathcal{X}$. Whenever a non-singleton regular cluster X is added to set $\mathcal{X}$, we define the parameter $d_X = \min\left\{\frac{|X^0| \cdot \eta}{\Delta}, d\right\}$. We may sometimes refer to d_X as d_{X^0}. Next, we partition all clusters that are ever added to $\mathcal{X}$ into *leaf* and *non-leaf* clusters.

Definition 5.4 (Leaf and Non-Leaf Clusters) *Let X be a cluster that was just added to $\mathcal{X}$. We say that X is a* leaf cluster*, if $|X^0| \leq \frac{\Delta \cdot (\log N)^{128}}{\eta}$. If X is not a leaf cluster, then we call it a* non-leaf *cluster.*

Recall that, if U is a special cluster, then $|U^0| > 2(n - |B^0|) \cdot (\log N)^{128}$. Since $\Delta \leq n - |B^0|$ from the definition of the s-t-SP problem, we get that a special cluster may not be a leaf cluster.

Recall that in general, once a cluster X is added to the collection $\mathcal{X}$ of clusters of an $\mathcal{ATO}$, it must remain in $\mathcal{X}$ throughout the algorithm, though we may delete some vertices from X. A new cluster Y may only be added to the set $\mathcal{X}$ if it is a subset of some cluster X that currently lies in $\mathcal{X}$. When we describe our algorithm for maintaining the $\mathcal{ATO}$, it will be convenient for us to sometimes remove a cluster X from the set $\mathcal{X}$, and then immediately reinsert it into $\mathcal{X}$. This is only done for clarity of exposition and convenience of analysis, and we do not need to do so in order to maintain the $\mathcal{ATO}$. Additionally, we will sometimes replace a cluster $X \in \mathcal{X}$ with a new collection of clusters, each of which contains a single vertex of X. In such a case we could simply identify X with one of the new clusters and implement this update to $\mathcal{X}$ via a sequence of cluster splitting operations, though for clarity of exposition we will treat all newly inserted clusters as new clusters, and we will think of X as being removed from $\mathcal{X}$.

Next, we describe an algorithm for maintaining each cluster X that is added to $\mathcal{X}$. We employ different algorithms for leaf clusters, non-leaf regular clusters, and the special cluster, if it is defined. We now describe each of the three algorithms in turn.

5.1.1 Leaf Clusters

Let X be a leaf cluster that was just added to $\mathcal{X}$. If X is a singleton cluster, then no further processing of this cluster is needed. Assume now that X is a non-singleton cluster. We immediately remove X from $\mathcal{X}$, and instead add, for every vertex $v \in X$, a new singleton cluster $\{v\}$ to $\mathcal{X}$. We assign, to each such new cluster $\{v\}$, an interval $I_{\{v\}} \subseteq I_X$ of length 1, so that all intervals corresponding to the vertices of $X \cap L$ appear before all intervals corresponding to the vertices of $X \cap R$. Notice that the only new right-to-left edges that are introduced by this transformation are the edges of $E_H(X \cap R, X \cap L)$, which are all special edges. The number of such edges is $|X \cap R| = |X \cap L| \leq |X|$. We say that these new right-to-left edges are *bad edges*, and that they are *owned* by the leaf cluster X. We denote the set of these edges by $E^{\mathsf{bad}}(X)$. From the above discussion, $|E^{\mathsf{bad}}(X)| \leq |X|$, and each edge $e \in E^{\mathsf{bad}}(X)$ has $\operatorname{span}(e) \leq |X|$.

5.1.2 Non-Leaf Regular Clusters

Whenever a new non-leaf regular cluster X is added to $\mathcal{X}$, we initialize Algorithm $\mathcal{A}$ for the r-restricted $\mathsf{MaintainCluster}$ problem on graph $H[X^0]$, with parameters Δ and η that remain unchanged, parameter d replaced with d_X, and parameter N replaced with N'. We denote this instantiation of Algorithm $\mathcal{A}$ by $\mathcal{A}(X)$. We now verify that $(H[X^0], \Delta, \eta, d_X, N')$ is indeed a valid input instance for the r-restricted $\mathsf{MaintainCluster}$ problem, in the following simple but technical observation, whose proof is deferred to Section B.1 of Appendix.

Observation 5.5 *$(H[X^0], \Delta, \eta, d_X, N')$ is a valid input to the r-restricted* $\mathsf{MaintainCluster}$ *problem.*

Recall that Algorithm $\mathcal{A}(X)$ may, at any time, produce a strongly well-structured cut (Y, Z) in $H[X]$ of sparsity $\Phi_{H[X]}(Y, Z) \leq \frac{(\log N')^{64}}{d_X}$. Below, we provide an algorithm for processing such a cut. But before we do so, we provide the remaining details for processing cluster X.

Recall that the algorithm for the s-t-SP problem consists of at most Δ iterations. At the beginning of every iteration i, we will compute, for every non-leaf regular cluster $X \in \mathcal{X}$, a pair $x, y \in V(X)$ of vertices (where possibly $x = y$). The corresponding algorithm $\mathcal{A}(X)$ is then required to compute a simple path P_X connecting x to y in graph $H[X]$, whose length is at most d_X. We will then process the resulting paths P_X for all non-leaf clusters $X \in \mathcal{X}$, in order to obtain a short s-t path P_i in H, as required. For each non-leaf cluster $X \in \mathcal{X}$, we will ensure that $E(P_i) \cap E(H[X]) \subseteq E(P_X)$. Following that, the lengths of some special edges on path P_i may be doubled; we assume that our algorithm is given the list of all such edges. In particular, for each non-leaf regular cluster $X \in \mathcal{X}$, we are given a subset of the special edges on path P_X, whose lengths must be doubled. From the definition of the s-t-SP problem, we are guaranteed that, for every non-leaf regular cluster $X \in \mathcal{X}$, for every special edge $e \in E(H[X])$, if $\tau' > \tau$ are two times at which the length of e is doubled, then the number of paths P_X that Algorithm $\mathcal{A}(X)$ returned in response to queries during the time interval $(\tau, \tau']$, which contain e, is at least η, so Property P1 holds for the instance of the $\mathsf{MaintainCluster}$ problem associated with X. If Algorithm $\mathcal{A}(X)$ ever returns "FAIL", then we terminate our algorithm for the s-t-SP problem, and return "FAIL" as well.

Lastly, whenever, for a non-leaf regular cluster $X \in \mathcal{X}$, $|X|$ falls below $\frac{|X^0|}{2}$, we terminate Algorithm $\mathcal{A}(X)$, and we say that cluster X is *destroyed*. We then think of the resulting set X of vertices as a new cluster that was just added to $\mathcal{X}$, and treat it as such (so we think that, once cluster X is destroyed, it is deleted from $\mathcal{X}$, and instead a new cluster, that is identical to X, is inserted into $\mathcal{X}$). It now remains to describe the algorithm to process sparse cuts produced by Algorithm $\mathcal{A}(X)$ for non-leaf regular clusters $X \in \mathcal{X}$.

Processing a Cut (Y, Z) in Cluster X

We assume that we are given a non-leaf regular cluster $X \in \mathcal{X}$, and a strongly well-structured cut (Y, Z) in $H[X]$ of sparsity $\Phi_{H[X]}(Y, Z) \leq \frac{(\log N')^{64}}{d_X}$. We denote by $J \subseteq Y$ and $J' \subseteq Z$ the sets of vertices that serve as endpoints of the edges of $E_H(Y, Z)$. Since the edges of $E_H(Y, Z)$ are special edges, it must be the case that $J \subseteq R$ and $J' \subseteq L$. Denote $Y' = Y \setminus J$ and $Z' = Z \setminus J'$. We start with the following simple observation, whose proof is deferred to Section B.2 of Appendix.

Observation 5.6 *Each of the graphs $H[Y'], H[Z']$ is a perfect well-structured graph.*

Assume first that $w(Y) \leq w(Z)$. We update the set X of vertices, by deleting the vertices of $Y \cup J'$ from it; in other words, we set $X = Z'$. We then add Y' as a new cluster to $\mathcal{X}$, and additionally, for every vertex $v \in J \cup J'$, we add a new singleton cluster $\{v\}$ to $\mathcal{X}$. We define the intervals associated with the new clusters, as follows. We assign, to every vertex $v \in J' \cup J$ an interval $I_{\{v\}} \subseteq I_X$ of length 1, and additionally we define two intervals: $I_{Y'} \subseteq I_X$ of length $|Y'|$ and $I_{Z'} \subseteq I_X$ of length $|Z'|$, so that all resulting intervals $I_{Y'}, I_{Z'}, \left\{I_{\{v\}}\right\}_{v \in J \cup J'}$ are mutually disjoint, and the intervals are ordered as follows: we place the intervals corresponding to the vertices of J' first, followed by interval $I_{Z'}$, then $I_{Y'}$, and lastly the intervals corresponding to the vertices of J. Since we have set $X = Z'$, we let $I_X = I_{Z'}$. From Observation 5.6, Invariant I1 continues to hold. We now show that Invariant I3 continues to hold, in the following observation, whose proof is deferred to Section B.3 of Appendix.

Observation 5.7 *If $e = (u, v)$ is a right-to-left edge that was added due to the above transformation of $\mathcal{ATO}$, then $u \in J$ and $v \in J'$ holds, and so $e \in E_H(Y, Z)$, and e is a special edge.*

Since we have defined a separate singleton cluster for every vertex of $J \cup J'$, from Observation 5.7, Invariant I3 continues to hold. Moreover, the only right-to-left edges that we have introduced during this update are the edges of $E_H(Y, Z)$. Notice that the $\text{span}(e)$ value of each such edge is at most $|X|$. We call all edges of $E_H(Y, Z)$ *bad edges*, and we say that they are *owned* by cluster X.

Recall that so far we have assumed that $w(Y) \leq w(Z)$. If $w(Y) > w(Z)$, then the algorithm remains the same, except that at the end we update cluster X by deleting the vertices of $Z \cup J$ from it; or equivalently, we set $X = Y'$. We also set $I_X = I_{Y'}$. This concludes the algorithm for processing the cut (Y, Z) of a non-leaf regular cluster $X \in \mathcal{X}$.

5.1.3 The Special Cluster

Recall that Case 1 happens if $|U^0| > 2(n - |B^0|) \cdot (\log N)^{128}$ holds. In this case, we let U be the special cluster of the $\mathcal{ATO}$, and we set $\gamma = |U^0| - 256(n - |B^0|)(\log N)^{64}$. We now show an algorithm for processing the special cluster U if Case 1 happened. We initialize Algorithm $\mathcal{A}'$ for the r-restricted $\mathsf{MaintainSpecialCluster}$ problem on graph $H[U^0]$, with parameters Δ and η that remain unchanged, parameter d replaced by parameter $d_U = \max\left\{(\log N')^{64}, \left(|U^0 \setminus B^0| + \Delta\right) \frac{\eta}{4\Delta \cdot 2^{\sqrt{\log N}}}\right\}$, and parameter N replaced with N'. We also initially set $\beta = B^0 \cap U^0$, and we ensure that, throughout the algorithm, $\beta = B \cap U$ holds. In other words, whenever a vertex of β is deleted from B or from U, it is also deleted from β.

We denote this instantiation of Algorithm $\mathcal{A}'$ by $\mathcal{A}'(U)$. We now verify that $(H[U^0], \beta, \Delta, \eta, d, N')$ is indeed a valid input instance for the r-restricted $\mathsf{MaintainSpecialCluster}$ problem, in the following simple observation, whose proof is deferred to Section B.4 of Appendix.

Observation 5.8 *$(H[U^0], \beta, \Delta, \eta, d_U, N')$ is a valid input to the r-restricted* $\mathsf{MaintainSpecialCluster}$ *problem.*

Recall that Algorithm $\mathcal{A}'(U)$ may, at any time, produce a strongly well-structured cut (Y, Z) in $H[U]$ of sparsity $\Phi_{H[U]}(Y, Z) \leq \frac{(\log N')^{64}}{d_U}$, such that $w(Y) \geq 1.8|\beta \cap Y|$ holds. Below, we provide an algorithm for processing such a cut. But before we do so, we provide the remaining details for processing cluster U. We will show later that, throughout the algorithm, $|\beta| \geq \frac{|B^0 \cap U^0|}{2}$ holds.

Recall that the algorithm for the s-t-SP problem consists of at most Δ iterations. At the beginning of every iteration i, we may compute a single vertex $x \in U$, that is sent as a query to Algorithm $\mathcal{A}'(U)$.

Algorithm $\mathcal{A}'(U)$ is then required to compute a simple path $P(x)$ connecting x to some vertex $y \in \beta$ in graph $H[U]$, whose length is at most d_U, such that no inner vertex of $P(x)$ lies in β (if $x \in \beta$, then we require that path $P(x)$ only consists of the vertex x). Our algorithm will then compute a short s-t path P_i in H, as required, so that $E(P_i) \cap E(H[U]) \subseteq E(P(x))$. Following that, the lengths of some special edges on path P_i may be doubled; we assume that our algorithm is given the list of all such edges. In particular, we are given a subset of the special edges on path $P(x)$, whose lengths must be doubled. Additionally, vertex y that serves as an endpoint of path $P(x)$ is deleted from set B and from set β. This ensures that the lengths of the special edges that are incident to vertices of β always remain 1. We note that in some of the iterations the algorithm for the s-t-SP problem may not ask any queries to algorithm $\mathcal{A}'(U)$, and in such iterations the lengths of the special edges of $H[U]$ are not doubled. From the definition of the s-t-SP problem, we are guaranteed that, for every special edge $e \in E(H[U])$, if $\tau' > \tau$ are two times at which the length of e is doubled, then the number of paths $P(x)$ that Algorithm $\mathcal{A}'(U)$ returned in response to queries during the time interval $(\tau, \tau']$, which contain e, is at least η, so Property P1 holds for the instance of the MaintainSpecialCluster problem associated with U. If Algorithm $\mathcal{A}'(U)$ ever returns "FAIL", then we terminate our algorithm for the s-t-SP problem, and return "FAIL" as well.

Finally, we provide an algorithm for processing the sparse cuts that Algorithm $\mathcal{A}'(U)$ may produce. Suppose Algorithm $\mathcal{A}'(U)$ returns a strongly well-structured cut (Y, Z) in $H[U]$ of sparsity $\Phi_{H[U]}(Y, Z) \leq \frac{(\log N')^{64}}{d_U}$, such that $w(Y) \geq 1.8|\beta \cap Y|$ holds. We denote by $J \subseteq Y$ and $J' \subseteq Z$ the sets of vertices that serve as endpoints of the edges of $E_H(Y, Z)$, and we let $Y' = Y \setminus J$ and $Z' = Z \setminus J'$. Using the same reasoning as in the proof of Observation 5.6, each of the graphs $H[Y'], H[Z']$ is a perfect well-structured graph. We will show below that $|Z'| \geq \gamma$ must hold. We delete the vertices of $Y \cup J'$ from U, so that $U = Z'$ now holds, and we add Y' as a new cluster to $\mathcal{X}$. Additionally, for every vertex $v \in J \cup J'$, we add a new singleton cluster $\{v\}$ to $\mathcal{X}$. We define the intervals associated with the new clusters as follows. We assign, to every vertex $v \in J' \cup J$ an interval $I_{\{v\}} \subseteq I_U$ of length 1, and additionally we define two intervals: $I_{Y'} \subseteq I_U$ of length $|Y'|$ and $I_{Z'} \subseteq I_U$ of length $|Z'| - \gamma + 1$, so that all resulting intervals $I_{Y'}, I_{Z'}, \left\{I_{\{v\}}\right\}_{v \in J \cup J'}$ are mutually disjoint, and the intervals are ordered as follows: we place the intervals corresponding to the vertices of J' first, followed by interval $I_{Z'}$, then $I_{Y'}$, and lastly the intervals corresponding to the vertices of J. Since we have set $U = Z'$, we let $I_U = I_{Z'}$. As before, Invariant I1 continues to hold. Using the same reasoning as in the proof of Observation 5.7, if $e = (u, v)$ is a right-to-left edge that was added due to the above transformation of $\mathcal{ATO}$, then $u \in J$ and $v \in J'$ holds, and so $e \in E_H(Y, Z)$, and e is a special edge. In particular, Invariant I3 continues to hold. Moreover, the only right-to-left edges that we have introduced during this update are the edges of $E_H(Y, Z)$. Notice that the span(e) value of each such edge is at most $|U^0| - \gamma + 1$. We call all edges of $E_H(Y, Z)$ *bad edges*, and we say that they are *owned* by cluster U. We denote by $E^{\mathsf{bad}}(U)$ the set of all bad edges that are owned by cluster U over the course of the algorithm. In the following observation we show that $|U| \geq \gamma$ holds throughout the algorithm, and we bound $|E^{\mathsf{bad}}(U)|$.

Observation 5.9 *Over the course of the algorithm, at most* $64|U^0 \setminus B^0| + 128\Delta \cdot (\log N)^{64} \leq 256(n - |B^0|)(\log N)^{64}$ *vertices are deleted from* U*, and so throughout the algorithm,* $|U| \geq \gamma$ *and* $|\beta| \geq \frac{|U^0 \cap B^0|}{2}$ *hold. Moreover,* $|E^{\mathsf{bad}}(U)| \leq \frac{\Delta \cdot 2^{6\sqrt{\log N}}}{\eta}$.

Proof: Recall that we denoted by $E^* \subseteq E^{\mathsf{spec}}$ the set of all special edges $e = (u, v)$, such that $u, v \in B$ holds at the end of the algorithm. Let $B^* \subseteq B$ the set of vertices containing all endpoints of the edges in E^*, and let $\tilde{U} = U^0 \setminus B^*$. For every vertex $v \in U^0$, we define a weight $\tilde{w}(v)$ as follows: if, at the end of the algorithm, $v \in U$, then $\tilde{w}(v)$ is the weight of v at the end of the algorithm; otherwise, $\tilde{w}(v)$ is the weight of v just before it is deleted from U. We use the following observation, whose proof is very similar to the proof of Observation 5.3, and deferred to Section B.5 of Appendix.

Observation 5.10 $\sum_{v\in\tilde{U}} \tilde{w}(v) \leq 8|U^0 \setminus B^0| + 8\Delta \cdot (\log N')^{64}$.

Consider now some cut (Y,Z) of U that Algorithm $\mathcal{A}'(U)$ produced. Recall that $|E_H(Y,Z)| \leq \frac{(\log N')^{64}}{d_U} \cdot w(Y)$, and that $w(Y) \geq 1.8|\beta \cap Y|$ holds. Since $B^* \cap Y \subseteq \beta \cap Y$, we get that $w(Y) \geq 1.8|B^* \cap Y|$. Since the weight of every vertex in B^* remains 1 throughout the algorithm, we get that $w(Y \setminus B^*) = w(Y) - |B^* \cap Y| \geq w(Y) - \frac{w(Y)}{1.8} \geq 0.44w(Y)$. In particular, $|Y \cap B^*| \leq \frac{w(Y)}{1.8} \leq \frac{w(Y\setminus B^*)}{1.8\cdot 0.44} \leq 1.3w(Y \setminus B^*)$. Moreover, $|Y| \leq w(Y) = w(Y \setminus B^*) + |Y \cap B^*| \leq 2.3w(Y \setminus B^*)$ holds. Recall that all vertices of Y, and also the endpoints of the special edges in $E_H(Y,Z)$ are deleted from cluster U following the cut (Y,Z). Therefore, the total number of vertices deleted from U following the cut (Y,Z) is at most $2|Y| \leq 4.6w(Y \setminus B^*) \leq 4.6\sum_{v\in Y\cap\tilde{U}} \tilde{w}(v)$. Lastly:

$$|E_H(Y,Z)| \leq \frac{(\log N')^{64}}{d_U} \cdot w(Y) \leq \frac{(\log N')^{64}}{d_U} \cdot 2.3w(Y \setminus B^*) \leq \frac{2.3 \cdot (\log N')^{64}}{d_U} \cdot \sum_{v\in Y\cap\tilde{U}} \tilde{w}(v).$$

Consider now all cuts $(Y_1,Z_1),\ldots,(Y_r,Z_r)$ that Algorithm $\mathcal{A}'(U)$ ever produced, in the order in which they were produced. For all $1 \leq i \leq r$, following the cut (Y_i,Z_i), the vertices of Y_i, along with some additional vertices, were deleted from U. From the above discussion, the total number of vertices deleted from U following the cut is bounded by $4.6\sum_{v\in Y_i\cap\tilde{U}} \tilde{w}(v)$, and $|E_H(Y_i,Z_i)| \leq \frac{2.3\cdot(\log N')^{64}}{d_U} \cdot \sum_{v\in Y\cap\tilde{U}} \cdot \sum_{v\in Y_i\cap\tilde{U}} \tilde{w}(v)$. We conclude that the total number of vertices deleted from U over the course of the algorithm is bounded by:

$$\begin{aligned}
4.6\sum_{i=1}^{r}\sum_{v\in Y_i\cap\tilde{U}} \tilde{w}(v) &\leq 4.6\sum_{v\in\tilde{U}} \tilde{w}(v)\\
&\leq 64|U^0 \setminus B^0| + 64\Delta \cdot (\log N')^{64}\\
&\leq 64|U^0 \setminus B^0| + 128\Delta \cdot (\log N)^{64}\\
&\leq 256(n - |B^0|)(\log N)^{64},
\end{aligned}$$

from Inequality 3. Since, at the beginning of the algorithm, $|U| > 2(n - |B^0|) \cdot (\log N)^{128}$ holds, and since $\gamma = |U^0| - 256(n - |B^0|)(\log N)^{64}$, we get that $|U| \geq \gamma$ holds throughout the algorithm.

Next, we show that, throughout the algorithm, $|\beta| \geq \frac{|U^0\cap B^0|}{2}$ holds. Recall again that $|U^0| > 2(n - |B^0|) \cdot (\log N)^{128}$, and that $|U^0 \setminus B^0| \leq n - |B^0|$. Therefore, $|U^0 \cap B^0| \geq |U^0| - |U^0 \setminus B^0| \geq (n - |B^0|) \cdot (\log N)^{128}$. Set β contains all vertices of $U^0 \cap B^0$, except for the vertices that were deleted from B (whose number is bounded by $\Delta \leq n - |B^0|$), and the vertices that were deleted from U (whose number is bounded by $256(n - |B^0|)(\log N)^{64}$). Therefore, throughout the algorithm, $|\beta| \geq |U^0 \cap B^0| - 257(n - |B^0|)(\log N)^{64}$ must hold. Since $|U^0 \cap B^0| \geq (n - |B^0|) \cdot (\log N)^{128}$, we get that $|\beta| \geq \frac{|U^0\cap B^0|}{2}$ holds throughout the algorithm.

Lastly, we get that:

$$\begin{aligned}
|E^{\mathsf{bad}}(U)| &\le \sum_{i=1}^{r} |E_H(Y_i, Z_i)| \\
&\le \frac{2.3 \cdot (\log N')^{64}}{d_U} \cdot \sum_{i=1}^{r} \sum_{v \in Y \cap \tilde{U}} \tilde{w}(v) \\
&\le \frac{10\Delta \cdot 2^{2\sqrt{\log N'}}}{(|U^0 \setminus B^0| + \Delta) \cdot \eta} \cdot \sum_{v \in \tilde{U}} \tilde{w}(v) \\
&\le \frac{10\Delta \cdot 2^{2\sqrt{\log N'}}}{(|U^0 \setminus B^0| + \Delta) \cdot \eta} \cdot \left(8|U^0 \setminus B^0| + 8\Delta(\log N')^{64}\right) \\
&\le \frac{\Delta \cdot 2^{6\sqrt{\log N}}}{\eta},
\end{aligned}$$

since $d_U = \max\left\{(\log N')^{64}, \left(|U^0 \setminus B^0| + \Delta\right) \frac{\eta}{4\Delta \cdot 2^{\sqrt{\log N}}}\right\} \ge \left(|U^0 \setminus B^0| + \Delta\right) \frac{\eta}{4\Delta \cdot 2^{\sqrt{\log N}}}$; we have also used Inequality 4. $\square$

5.1.4 Child Clusters, Descendant Clusters, and a Useful Bound

Let $\tilde{\mathcal{X}}$ denote the set of all clusters that ever belonged to $\mathcal{X}$ over the course of the algorithm. We partition $\tilde{\mathcal{X}}$ into three subsets: set $\tilde{\mathcal{X}}_0$ contains all singleton clusters, set $\tilde{\mathcal{X}}_1$ contains all non-singleton leaf clusters, and set $\tilde{\mathcal{X}}_2$ contains all non-leaf clusters. If the special cluster is defined, then we set $\tilde{\mathcal{X}}_2' = \tilde{\mathcal{X}}_2 \setminus \{U\}$, and otherwise we let $\tilde{\mathcal{X}}_2' = \tilde{\mathcal{X}}_2$.

Let E^{bad} be the set of all edges of G that ever served as right-to-left edges of the $\mathcal{ATO}$. For every cluster $X \in \tilde{\mathcal{X}}_1 \cup \tilde{\mathcal{X}}_2$, we denote by $E^{\mathsf{bad}}(X)$ the set of all bad edges that cluster X owns. Recall that, once an edge $e = (x, y)$ is added to the set $E^{\mathsf{bad}}(X)$ of edges, its initial value span(e) is bounded by $|X|$. From Invariant I3, once edge e was added to set E^{bad}, if Y, Y' were the clusters of $\mathcal{X}$ containing x and y respectively, then $Y = \{x\}$ and $Y' = \{y\}$ holds. In other words, when e was added to E^{bad}, Y and Y' were singleton clusters. Therefore, these clusters, and their corresponding intervals I_Y and $I_{Y'}$, remain unchanged for the remainder of the algorithm. In particular, the value span(e) also remains unchanged for the remainder of the algorithm.

Recall that, from Invariant I1, for every cluster $X \in \tilde{\mathcal{X}}_2'$, graph $H[X]$ remains a perfect well-structured graph for as long as X lies in $\mathcal{X}$. Recall also that we have defined weights of vertices for such graphs: for every vertex $v \in X$, its weight $w(v)$ is the length of the unique special edge incident to v in $H[X]$. As the algorithm progresses, the lengths of some special edges grow, and so do the weights of the vertices in $H[X]$. For every vertex $v \in X^0$, we denote by $w^0_X(v)$ the weight of v when cluster X was first added to set $\mathcal{X}$, and algorithm $\mathcal{A}(X)$ was initialized. If v belongs to set X when algorithm $\mathcal{A}(X)$ is terminated (e.g. because $|X|$ fell below $|X^0|/2$), then we let $w^*_X(v)$ denote the weight of v in graph $H[X]$ when algorithm $\mathcal{A}(X)$ terminated. Otherwise, we denote by $w^*_X(v)$ the weight of v just before it was deleted from X. We also denote $W^0(X) = \sum_{v \in X^0} w^0_X(v)$, and $W^*(X) = \sum_{v \in X^0} w^*_X(v)$. From Observation 5.3, the total length of all edges in E^{spec} at the end of the algorithm is bounded by $|E^{\mathsf{spec}}| + 2(n - |B^0|)\log^5 n \le 4n \log^5 n$, and so, for every cluster $X \in \tilde{\mathcal{X}}_2$:

$$W^0(X) \le W^*(X) \le 8n \log^5 n. \tag{6}$$

Recall that, at the beginning of the algorithm, $\mathcal{X}$ may contain at most one non-leaf cluster – the cluster U. A new non-leaf cluster X' may only be inserted into $\mathcal{X}$ in one of the following two cases:

(i) there is some existing non-leaf cluster $X \in \mathcal{X}$, for which $|X|$ fell below $|X^0|/2$, and $X' = X$; or (ii) there is some existing non-leaf cluster $X \in \mathcal{X}$, for which Algorithm $\mathcal{A}(X)$ or $\mathcal{A}'(X)$ just computed a sparse well-structured cut (Y, Z), and X' is contained in one of the sets Y or Z. In the latter case, the vertices of X' (along, possibly, with some additional vertices) are deleted from X. In either one of the above two cases, we say that X is the *parent-cluster* of X', and X' is the *child-cluster* of X. We can now naturally define a descendant-ancestor relationship between clusters: Cluster X' is a descendant-cluster of another cluster X, iff there is a sequence $X' = X_1, X_2, \ldots, X_r = X$ of $r \geq 1$ clusters, such that, for all $1 \leq i < r$, X_i is a child-cluster of X_{i+1}. If X' is a descendant-cluster of X, then X is the ancestor-cluster of X'. It is easy to verify that, for every pair $X_1, X_2 \in \tilde{\mathcal{X}}_2$ of clusters, either $X_1^0 \cap X_2^0 = \emptyset$, or $X_1^0 \subseteq X_2^0$ (in which case X_1 is a descendant cluster of X_2), or $X_2^0 \subseteq X_1^0$ (in which case X_2 is a descendant cluster of X_1). From the above discussion, if X_1 is a child-cluster of some cluster $X_2 \in \tilde{\mathcal{X}}'_2$, then either $|X_1^0| < |X_2^0|/2$; or, at the time when X_1 was created, $\sum_{v \in X_1} w(v) \leq \sum_{v \in X_2} w(v)/2$ held, following which the vertices of X_1 were deleted from X_2.

Consider now a pair $X_1, X_2 \in \tilde{\mathcal{X}}'_2$ of clusters, where X_2 is a descendant cluster of X_1. We say that X_2 is a *close descendant of* X_1 iff $|X_2^0| > |X_1^0|/2$. In the following claim, we bound the number of close descendants of every cluster in $\tilde{X}'_2$. The proof is somewhat technical and deferred to Section B.6 of Appendix.

Claim 5.11 *Every cluster $X \in \tilde{X}'_2$ has at most $64 \lceil \log n \rceil$ close descendants.*

For the sake of analysis, we need to bound $\sum_{X \in \tilde{\mathcal{X}}'_2} W^*(X)$. This bound will later be used both in order to bound the running time of the algorithm, and in order to bound $\sum_{e \in E^{\mathsf{bad}}} \mathrm{span}(e)$.

Observation 5.12 $\sum_{X \in \tilde{\mathcal{X}}'_2} W^*(X) \leq (n - |B^0|) \cdot (\log N)^{133}$.

Proof: Recall that for every cluster $X \in \tilde{\mathcal{X}}'_2$, $W^0(X) \leq 8n \log^5 n$ holds (see Inequality 6). For all $0 \leq i < \lceil \log n \rceil$ and $0 \leq j \leq 4 \lceil \log n \rceil$, we denote by $\mathcal{X}_{i,j}$ the set of all clusters $X \in \tilde{\mathcal{X}}'_2$ with $2^i \leq |X^0| < 2^{i+1}$ and $2^j \leq W^0(X) < 2^{j+1}$.

Fix a pair $0 \leq i < \lceil \log n \rceil$ and $0 \leq j \leq 4 \lceil \log n \rceil$ of indices, and consider the collection $\mathcal{X}_{i,j}$ of clusters. Notice that, if $X, X' \in \mathcal{X}_{i,j}$, and X' is a descendant-cluster of X, then X' is a close descendant of X. From Claim 5.11, for every cluster $X \in \mathcal{X}_{i,j}$, at most $64 \lceil \log n \rceil$ descendant clusters of X lie in $\mathcal{X}_{i,j}$. Using a simple greedy algorithm, we can then paratition the clusters in $\mathcal{X}_{i,j}$ into $z = 64 \lceil \log n \rceil + 1$ subsets $\mathcal{Y}_1, \ldots, \mathcal{Y}_z$, such that, for all $1 \leq z' \leq z$, for every pair $X, X' \in \mathcal{Y}_{z'}$ of clusters, neither of the two clusters is a descendant of the other. In order to do so, we consider the clusters $X \in \mathcal{X}_{i,j}$ one by one, in the non-decreasing order of $|X^0|$, breaking ties arbitrarily. When cluster X is considered, there must be some index $1 \leq z' \leq z$, such that no descendant of X lies in $\mathcal{Y}_{z'}$, and we add X to such a set $\mathcal{Y}_{z'}$. For all $1 \leq z' \leq z$, we now bound $\sum_{X \in \mathcal{Y}_{z'}} W^*(X)$ separately.

Fix any index $1 \leq z' \leq z$, and denote $\mathcal{Y}' = \left\{X^0 \mid X \in \mathcal{Y}_{z'}\right\}$. Since, for every pair X, X' of clusters in $\mathcal{Y}_{z'}$, neither is the descendant of the other, all sets in $\mathcal{Y}'$ are mutually disjoint.

Assume first that Case 2 happened, so the special cluster is undefined, and $|U^0| \leq 2(n - |B^0|) \cdot (\log N)^{128}$ holds. From the construction of the set U of vertices at the beginning of the algorithm, for every special edge $e \in E^{\mathsf{spec}}$, both endpoints of e lie in U^0. Therefore, $|E^{\mathsf{spec}}| \leq \frac{|U^0|}{2} \leq (n - |B^0|) \cdot (\log N)^{128}$. From Observation 5.3, the total length of all edges in E^{spec} at the end of the algorithm is bounded by $|E^{\mathsf{spec}}| + 2(n - |B^0|) \log^5 n \leq (n - |B^0|) \cdot (\log N)^{129}$. Since the clusters in $\mathcal{Y}'$ are mutually disjoint, it is easy to verify that $\sum_{X \in \mathcal{Y}_{z'}} W^*(X) \leq 2(n - |B^0|) \cdot (\log N)^{129}$.

Next, we consider Case 1, where the special cluster is defined. Let U^* denote the set U of vertices at the end of the algorithm. Notice that U^* is disjoint from all sets of vertices in $\mathcal{Y}'$. For every vertex

$v \in U^*$, let $w^*(v)$ be the weight of v at the end of the algorithm, and let $W^*(U) = \sum_{v \in U^*} w^*(v)$. From Observation 5.3, the total length of all edges in E^{spec} at the end of the algorithm is bounded by $|E^{\mathsf{spec}}| + 2(n - |B^0|)\log^5 n$, and so:

$$\sum_{X \in \mathcal{Y}_{z'}} W^*(X) + W^*(U) \le 2|E^{\mathsf{spec}}| + 4(n - |B^0|)\log^5 n. \tag{7}$$

Recall that all vertices that serve as endpoints of the edges in E^{spec} lie in U^0. From Observation 5.9, $|U^0 \setminus U^*| \le 256(n - |B^0|) \cdot (\log N)^{64}$. Therefore, $|U^*| \ge 2|E^{\mathsf{spec}}| - 256(n - |B^0|) \cdot (\log N)^{64}$, and $W^*(U) \ge |U^*| \ge 2|E^{\mathsf{spec}}| - 256(n - |B^0|) \cdot (\log N)^{64}$. Combining this with Inequality 7, we get that:

$$\sum_{X \in \mathcal{Y}_{z'}} W^*(X) \le 2|E^{\mathsf{spec}}| + 4(n - |B^0|)\log^5 n - W^*(U) \le 260(n - |B^0|) \cdot (\log N)^{64} \le 2(n - |B^0|) \cdot (\log N)^{129}.$$

To conclude, we have shown that, for every index $1 \le z' \le z$:

$$\sum_{X \in \mathcal{Y}_{z'}} W^*(X) \le 2(n - |B^0|) \cdot (\log N)^{129}.$$

Summing this up over all indices $1 \le z' \le z$, and recalling that $z = 64\lceil \log n \rceil + 1 \le 128 \log n$, we get that:

$$\sum_{X \in \mathcal{X}_{i,j}} W^*(X) \le 256(n - |B^0|) \cdot (\log N)^{130}.$$

Lastly, by summing up the above inequality over all indices $0 \le i < \lceil \log n \rceil$ and $0 \le j \le 4\lceil \log n \rceil$, we get that:

$$\sum_{X \in \tilde{\mathcal{X}}_2'} W^*(X) \le 2^{12}(n - |B^0|) \cdot (\log N)^{132} \le (n - |B^0|) \cdot (\log N)^{133}.$$

□

We obtain the following corollary of Observation 5.12, that allows us to bound the total span of all bad edges owned by the clusters in $\tilde{\mathcal{X}}_2'$.

Corollary 5.13 $\sum_{X \in \tilde{\mathcal{X}}_2'} \sum_{e \in E^{\mathsf{bad}}(X)} \mathrm{span}(e) \le \frac{(n - |B^0|) \cdot \Delta \cdot 2^{4\sqrt{\log N}}}{\eta}$.

Proof: Consider a cluster $X \in \mathcal{X}_2'$, and let (Y, Z) be a cut that Algorithm $\mathcal{A}(X)$ returned. Recall that $|E_H(Y, Z)| \le \frac{(\log N')^{64}}{d_X} \cdot \min\{w(Y), w(Z)\}$. If $w(Y) \le w(Z)$, then let $Z' = Y$, and otherwise let $Z' = Z$. Then $|E_H(Y, Z)| \le \frac{(\log N')^{64}}{d_X} \cdot w(Z') \le \frac{(\log N')^{64}}{d_X} \cdot \sum_{v \in Z'} w_X^*(v)$ holds. Moreover, all vertices of Z' are deleted from X following the cut (Y, Z), and the edges of $E_H(Y, Z)$ are added to the set $E^{\mathsf{bad}}(X)$ of bad edges that X owns. We assign to every vertex $v \in Z'$ a *charge* of $\frac{(\log N')^{64}}{d_X} \cdot w_X^*(v)$ units, so that the total charge assigned to the vertices of Z' is at least $|E_H(Y, Z)|$. It is then easy to see that, at the end of Algorithm $\mathcal{A}(X)$, the total number of edges in $E^{\mathsf{bad}}(X)$ is bounded by the total charge to all vertices of X^0, which, in turn, is bounded by:

$$\frac{(\log N')^{64}}{d_X} \cdot \sum_{v \in X^0} w_X^*(v) = \frac{(\log N')^{64}}{d_X} \cdot W^*(X).$$

Next, we show that $d_X \geq \frac{|X^0|\cdot\eta}{\Delta\cdot 2^{2\sqrt{\log N}}}$. Indeed, recall that $d = \frac{(n-|B^0|)\cdot\eta}{4\Delta\cdot 2^{\sqrt{\log N}}}$ and so:

$$d_X = \min\left\{\frac{|X^0|\cdot\eta}{\Delta}, d\right\} = \min\left\{\frac{|X^0|\cdot\eta}{\Delta}, \frac{(n-|B^0|)\cdot\eta}{4\Delta\cdot 2^{\sqrt{\log N}}}\right\}. \tag{8}$$

If Case 2 happened, then $|U^0| \leq 2(n-|B^0|)\cdot(\log N)^{128}$, and, since $X^0 \subseteq U^0$, we get that $|X^0| \leq 2(n-|B^0|)\cdot(\log N)^{128}$. If Case 1 happened, then, from Observation 5.9, $|X^0| \leq 256(n-|B^0|)(\log N)^{64}$. Therefore, in any case, $|X^0| \leq 2(n-|B^0|)\cdot(\log N)^{128}$ must hold. Substituting the bound $n-|B^0| \geq \frac{|X^0|}{2(\log N)^{128}}$ into Inequality 8, we get that $d_X \geq \frac{|X^0|\cdot\eta}{\Delta\cdot 2^{2\sqrt{\log N}}}$. Altogether, we get that:

$$|E^{\mathsf{bad}}(X)| \leq \frac{(\log N')^{64}}{d_X}\cdot W^*(X) \leq \frac{\Delta\cdot 2^{2\sqrt{\log N}}\cdot(\log N')^{64}}{|X^0|\cdot\eta}\cdot W^*(X) \leq \frac{\Delta\cdot 2^{3\sqrt{\log N}}}{|X^0|\cdot\eta}\cdot W^*(X).$$

Since for every edge $e \in E^{\mathsf{bad}}(X)$, $\text{span}(e) \leq |X^0|$, we get that:

$$\sum_{e\in E^{\mathsf{bad}}(X)} \text{span}(e) \leq |X^0|\cdot|E^{\mathsf{bad}}(X)| \leq \frac{\Delta\cdot 2^{3\sqrt{\log N}}}{\eta}\cdot W^*(X).$$

Finally, from Observation 5.12:

$$\begin{aligned}\sum_{X\in\tilde{\mathcal{X}}_2'}\sum_{e\in E^{\mathsf{bad}}(X)} \text{span}(e) &\leq \frac{\Delta\cdot 2^{3\sqrt{\log N}}}{\eta}\cdot\sum_{X\in\tilde{\mathcal{X}}_2'} W^*(X)\\ &\leq \frac{(n-|B^0|)\cdot\Delta\cdot 2^{3\sqrt{\log N}}\cdot(\log N)^{133}}{\eta}\\ &\leq \frac{(n-|B^0|)\cdot\Delta\cdot 2^{4\sqrt{\log N}}}{\eta}.\end{aligned}$$

□

5.1.5 Bounding the Running Time of Part 1 and Probability of Failure

Recall that, for every cluster $X \in \tilde{\mathcal{X}}_2'$, the total running time of Algorithm $\mathcal{A}(X)$ is bounded by:

$$\begin{aligned}&c_1\cdot(W^0(X))^2\cdot 2^{c_2\sqrt{\log N'}}\cdot(\log N')^{16c_2r} + c_1\cdot W^0(X)\cdot\Delta\cdot\log^4 N'\\ &\qquad\leq c_1\cdot W^0(X)\cdot(W^0(X)+\Delta)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+6}\cdot\left(1+\frac{1}{64c_2r}\right)^{16c_2r}\\ &\qquad\leq c_1\cdot W^0(X)\cdot(W^0(X)+\Delta)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+7}.\end{aligned}$$

(We have used the fact that, from Inequality 4, $2^{\sqrt{\log N'}} \leq 2^{\sqrt{\log N}}\cdot\log^6 N$, and from Inequality 3, $\log N' \leq \left(1+\frac{1}{64c_2r}\right)\cdot\log N$. Additionally, we used the assumption that N is sufficiently large.).

Summing up over all clusters $X \in \tilde{\mathcal{X}}_2'$, from Observation 5.12, we get that the total running time of all Algorithm $\mathcal{A}(X)$ for clusters $X \in \tilde{\mathcal{X}}_2'$ is bounded by:

$$\begin{aligned}\sum_{X\in\tilde{\mathcal{X}}_2'} & c_1\cdot W^0(X)\cdot(W^0(X)+\Delta)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+7}\\ &\le\left(\Delta\cdot\sum_{X\in\tilde{\mathcal{X}}_2'}W^0(X)+\left(\sum_{X\in\tilde{\mathcal{X}}_2'}W^0(X)\right)^2\right)\cdot c_1\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+7}\\ &\le c_1\cdot(n-|B^0|)^2\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+273}\\ &\le c_1\cdot n\cdot(n-|B^0|)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+273}.\end{aligned}$$

(We have used the fact that $\Delta\le n-|B^0|$ from the problem definition).

Assume now that Case 1 happened. In this case, the running time of Algorithm $\mathcal{A}'(U)$ is bounded by:

$$\begin{aligned}c_1 & \cdot W^0(U)\cdot(W^0(U)-|\beta|+\Delta)\cdot 2^{c_2\sqrt{\log N'}}\cdot(\log N')^{16c_2r}\\ &\le c_1\cdot W^0(U)\cdot(W^0(U)-|\beta|+\Delta)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+6}\cdot\left(1+\frac{1}{64c_2r}\right)^{16c_2r}\\ &\le c_1\cdot W^0(U)\cdot(W^0(U)-|\beta|+\Delta)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+7}\\ &\le 2c_1\cdot n\cdot(n-|B^0|)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+7}.\end{aligned}$$

(As before, we have used the fact that, from Inequality 4, $2^{\sqrt{\log N'}}\le 2^{\sqrt{\log N}}\cdot\log^6 N$, and from Inequality 3, $\log N'\le\left(1+\frac{1}{64c_2r}\right)\cdot\log N$. Additionally, since, at the beginning of the algorithm, the length of every special edge is 1, $W^0(U)=|U^0|\le n$. From the problem definition, $\Delta\le n-|B^0|$, and it is easy to see that $W^0(U)-|\beta|=|U^0|-|\beta|\le n-|B^0|$ must hold).

Additional time that is required in order to maintain the $\mathcal{ATO}$ is asymptotically bounded by the total number of edges that are ever present in H, which, from the problem definition, is bounded by $O(|E(H)|)\le O(n\cdot(n-|B^0|))$. Overall, since we can assume that c_2 is a large enough constant, the running time of Part 1 of the algorithm is bounded by:

$$c_1\cdot n\cdot(n-|B^0|)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+c_2}.$$

Notice that, since $N\ge n$ is large enough, the above running time is also bounded by N^3. Therefore, the total number of clusters that ever lie in $\mathcal{X}$ is also bounded by N^3. For each regular non-leaf cluster $X\in\tilde{\mathcal{X}}$, the probability that Algorithm $\mathcal{A}(X)$ returns "FAIL" is bounded by $\frac{1}{(N')^4}\le\frac{1}{N^4}$. If Case 1 happens, then the probability that Algorithm $\mathcal{A}'(U)$ returns "FAIL" is bounded by $\frac{1}{4}$. From the union bound, the total probability that our algorithm ever returns "FAIL" is bounded by $\frac{1}{2}$, as required.

5.1.6 Bounding the Total Span of Bad Edges

Recall that, from Corollary 5.13, $\sum_{X\in\tilde{\mathcal{X}}_2'}\sum_{e\in E^{\mathsf{bad}}(X)}\mathrm{span}(e)\le\frac{(n-|B^0|)\cdot\Delta\cdot 2^{4\sqrt{\log N}}}{\eta}$.

If Case 1 happened, then, from Observation 5.9, $|E^{\mathsf{bad}}(U)|\le\frac{\Delta\cdot 2^{6\sqrt{\log N}}}{\eta}$. It is easy to verify that, for every edge $e\in E^{\mathsf{bad}}(U)$, $\mathrm{span}(e)\le|U^0|-\gamma\le 256(n-|B^0|)(\log N)^{64}$, since $\gamma=|U^0|-256(n-|B^0|)(\log N)^{64}$. Therefore, $\sum_{e\in E^{\mathsf{bad}}(U)}\mathrm{span}(e)\le\frac{\Delta\cdot 2^{6\sqrt{\log N}}}{\eta}\cdot 256(n-|B^0|)(\log N)^{64}\le\frac{(n-|B^0|)\cdot\Delta\cdot 2^{7\sqrt{\log N}}}{\eta}$.

Lastly, consider a leaf cluster $X \in \tilde{\mathcal{X}}_1$, and recall that $|X^0| \le \frac{\Delta\cdot(\log N)^{128}}{\eta}$. Recall that $|E^{\mathsf{bad}}(X)| \le |X^0|$, and so $\sum_{e\in E^{\mathsf{bad}}(X)} \mathrm{span}(e) \le |X^0|^2 \le |X^0| \cdot \frac{\Delta\cdot(\log N)^{128}}{\eta}$.

Note that all clusters in $\tilde{\mathcal{X}}_1$ are mutually disjoint, and, for each such cluster X, $X^0 \subseteq U^0$ holds. If Case 2 happened, then $|U^0| \le 2(n-|B^0|)\cdot(\log N)^{128}$, and so $\sum_{X\in\tilde{\mathcal{X}}_1}|X^0| \le |U^0| \le 2(n-|B^0|)\cdot(\log N)^{128}$. Otherwise, from Observation 5.9, $\sum_{X\in\tilde{\mathcal{X}}_1}|X^0| \le 256(n-|B^0|)(\log N)^{64}$. Therefore, in either case, $\sum_{X\in\tilde{\mathcal{X}}_1}|X^0| \le 2(n-|B^0|)\cdot(\log N)^{128}$ holds, and so overall:

$$\sum_{X\in\tilde{\mathcal{X}}_1}\sum_{e\in E^{\mathsf{bad}}(X)} \mathrm{span}(e) \le \frac{\Delta\cdot(\log N)^{128}}{\eta}\cdot\sum_{X\in\tilde{\mathcal{X}}_1}|X^0| \le \frac{2(n-|B^0|)\cdot\Delta\cdot(\log N)^{256}}{\eta}.$$

Summing up over the three types of clusters, we get that:

$$\sum_{e\in E^{\mathsf{bad}}} \mathrm{span}(e) \le \frac{4(n-|B^0|)\cdot\Delta\cdot 2^{7\sqrt{\log N}}}{\eta}. \tag{9}$$

5.2 Part 2: Solving DAG-like SSSP on the Contracted Graph

The second part of our algorithm, we will define a dynamic graph $\hat{H}$ that our algorithm will maintain, which in turn will be used to define an instance of the DAG-like SSSP problem (see Section 2.4). We will then apply the algorithm for DAG-like SSSP from Theorem 2.7 to this instance of the problem. We start by defining the dynamc graph $\hat{H}$, and by providing an algorithm for maintaining it.

Definition of the Dynamic Graph $\hat{H}$

Consider any time τ during the time horizon $\mathcal{T}$, the graph H and the collection $\mathcal{X}$ of clusters at time τ. We say that a cluster $X \in \mathcal{X}$ is *distinguished*, if it contains some vertex that currently lies in B.

Graph $\hat{H}$ at time τ is defined as follows. We start with graph H, and, for every cluster $X \in \mathcal{X}$ that is not a distinguished cluster, we contract all vertices in X into a single vertex v_X (that we refer to as a *supernode*). Additionally, all vertices that lie in distinguished clusters, together with the vertex t, are contracted into a single supernode, that for convenience we continue to denote by t. We delete all loops but keep parallel edges. We also denote the supernode v_S corresponding to cluster $S=\{s\}$ by s. Consider now any edge e in the resulting graph. This edge corresponds to some edge in H whose endpoints lie in different clusters; we do not distinguish between these two edges. We replace every edge $e=(v_X,v_Y)$ in the resulting graph by a collection $J(e)$ of $O(\log^2 n)$ parallel edges (v_X,v_Y), as follows. Let 2^j be the smallest power of 2 that is greater than $\mathrm{skip}(e)$. If e is a regular edge in H, then set $J(e)$ contains, for all $j \le j' \le \lceil\log n\rceil$, a copy $e_{j'}$ of e, whose length is $\hat{\ell}(e_{j'})=1$, and weight $w(e_{j'})=2^{j'}$. Assume now that e is a special edge of H, and denote its length in H by $\ell(e)=2^i$. For all $i \le i' \le \lceil\log\Lambda\rceil$ and $j \le j' \le \lceil\log n\rceil$, set $J(e)$ contains a copy $e_{i',j'}$ of e with length $\hat{\ell}(e_{i',j'})=2^{i'}$ and weight $w(e_{i',j'})=2^{j'}$. This completes the definition of graph $\hat{H}$.

Consider now some vertex $v \in V(\hat{H})\setminus\{t\}$, and some integer $0 \le i \le \lceil\log n\rceil$. Consider the set of vertices $U_i(v)=\left\{u\in V(\hat{H}) \mid \exists e=(v,u)\in E(\hat{H}) \text{ with } w(e)=2^i\right\}$. Notice that set $U_i(v)$ may only contain the nodes s,t, and all nodes $v_Y \in V(\hat{H})$, such that some edge e of H connects a vertex of X to a vertex of Y, and has value $\mathrm{skip}(e) \le 2^i$. In the latter case, the distance between intervals I_X and I_Y in the $\mathcal{ATO}$ is at most 2^i, and so at most 2^i other intervals of the $\mathcal{ATO}$ may lie between them. Therefore, it is immediate to verify that $|U_i(v)| \leq 2^{i+2}$ always holds, establishing Property S1 from the definition of the DAG-like SSSP problem. Let n' be the total number of vertices that are ever present in the dynamic graph $\hat{H}$. We bound n' in the following simple observation.

Observation 5.14 $n' \leq 2(n - |B^0|) + 2$.

Proof: Let B^* be the set of vertices that lie in B at the end of the algorithm. Clearly, $n' \leq n - |B^*| + 2$. On the other hand, since the algorithm consists of $\Delta \leq n - |B^0|$ iterations, and in every iteration at most one vertex is deleted from B, we get that $|B^*| \geq |B^0| - \Delta \geq 2|B^0| - n$. Therefore, $n' \leq n - |B^*| + 2 \leq 2(n - |B^0|) + 2$. □

Maintaining the Graph $\hat{H}$

At the beginning of the algorithm, we construct the initial graph $\hat{H}$ from graph H as described above. It is easy to verify that this can be done in time $O(|E(H)| \cdot \log^2 n) \leq O(n(n - |B^0|) \log^2 n)$.

Next, we show an algorithm that maintains the graph $\hat{H}$, as graph H and the collection $\mathcal{X}$ of clusters evolve over time. We will ensure that the only updates that $\hat{H}$ undergoes are edge deletions and vertex splitting, as is required from the input to the DAG-like SSSP problem.

Recall that the only updates that graph H undergoes are (i) doubling of the lengths of its edges; and (ii) deletion of edges incident to s and t. Additionally, some vertices may be deleted from set B.

If an edge e that is incident to s or t is deleted from H, then we delete all copies of e from $\hat{H}$. If the length of some edge e is doubled, and the initial length of edge e was 2^i, then we delete all copies of e whose length is 2^i from graph $\hat{H}$.

Assume now that some vertex v is deleted from set B. Let $X \in \mathcal{X}$ be the cluster that contains v. If X remains a distinguished cluster, then no additional updates to graph $\hat{H}$ are required. Otherwise, we perform a *vertex splitting* operation, that we apply to supernode t, to construct a new supernode v_X. The corresponding collection E' of edges is defined as follows. Let $e = (a, b)$ be an edge of H, with exactly one endpoint in X; assume w.l.o.g. that $a \in X$ and $b \in Y$, where $Y \in \mathcal{X}$ is some cluster. Let 2^j be the smallest power of 2 that is greater than $\text{skip}(e)$. Assume first that e is a regular edge in H. Notice that, if Y is not a distinguished cluster, then, for all $j \leq j' \leq \lceil \log n \rceil$, graph $\hat{H}$ already contains an edge (t, v_Y), that is a copy of e, whose length is 1, and weight is $2^{j'}$. We add to E' a copy (v_X, v_Y) of e, whose length is 1, and weight is $2^{j'}$. Once the vertex splitting operation is completed, we will delete from $\hat{H}$ all copies of e that are incident to t. If Y is a distinguished cluster, then, for all $j \leq j' \leq \lceil \log n \rceil$, we insert a copy (v_X, t) of the edge e into E', whose length is 1 and weight is $2^{j'}$. Assume now that e is a special edge of H, and denote its length in H by $\ell(e) = 2^i$. Consider a pair $i \leq i' \leq \lceil \log n \rceil$ and $j \leq j' \leq \lceil \log n \rceil$ of indices. As before, if Y is not a distinguished cluster, then graph $\hat{H}$ already contains an edge (t, v_Y), that is a copy of e, whose length is $2^{i'}$, and weight is $2^{j'}$. We add to E' a copy (v_X, v_Y) of e, whose length is $2^{i'}$, and weight is $2^{j'}$. Once the vertex splitting operation is completed, we will delete from $\hat{H}$ all copies of e that are incident to t. If Y is a distinguished cluster, then we insert a copy (v_X, t) of the edge e into E', whose length is $2^{i'}$ and weight is $2^{j'}$. The case where $b \in X$ and $a \notin X$ is treated similarly.

Lastly, it remains to consider updates to the clusters in $\mathcal{X}$. All such updates can be implemented by the following two operations: (i) given a cluster $X \in \mathcal{X}$, and a subset $X' \subseteq X$ of vertices, insert X' into $\mathcal{X}$ (we call this update *cluster splitting*); and (ii) given a cluster $X \in \mathcal{X}$ and a vertex $x \in X$, delete x from X (we call this update *vertex deletion*).

Consider first a cluster splitting operation, that is applied to cluster $X \in \mathcal{X}$, through which a new cluster $X' \subseteq X$ is inserted into $\mathcal{X}$. Assume first that X is not a distinguished cluster. We apply a vertex splitting operation to supernode v_X, inserting a new supernode $v_{X'}$ into $\hat{H}$. The corresponding set E' of edges is defined as follows. Let $e = (a, b)$ be an edge of H, with exactly one endpoint in X'; assume w.l.o.g. that $a \in X'$ and $b \in Y$, where $Y \in \mathcal{X}$ is some cluster. Let 2^j be the smallest power of 2 that is greater than $\text{skip}(e)$. Assume first that e is a regular edge in H. Notice that, if $Y \neq X$, then, for all $j \leq j' \leq \lceil \log n \rceil$, graph $\hat{H}$ already contains an edge (v_X, v_Y), that is a copy of e, whose length is 1, and weight is $2^{j'}$. We add to E' a copy $(v_{X'}, v_Y)$ of e, whose length is 1, and weight is $2^{j'}$. If $Y = X$, then, for all $j \leq j' \leq \lceil \log n \rceil$, we insert a copy $(v_{X'}, v_X)$ of the edge e into E', whose length is 1 and weight is $2^{j'}$. Assume now that e is a special edge of H, and denote its length in H by $\ell(e) = 2^i$. Consider a pair $i \leq i' \leq \lceil \log n \rceil$ and $j \leq j' \leq \lceil \log n \rceil$ of indices. As before, if $Y \neq X$, then graph $\hat{H}$ already contains an edge (v_X, v_Y), that is a copy of e, whose length is $2^{i'}$, and weight is $2^{j'}$. We add to E' a copy $(v_{X'}, v_Y)$ of e, whose length is $2^{i'}$, and weight is $2^{j'}$. If $Y = X$, then we insert a copy $(v_{X'}, v_X)$ of the edge e into E', whose length is $2^{i'}$ and weight is $2^{j'}$. The case where $b \in X'$ and $a \notin X'$ is treated similarly.

If X' is a distinguished cluster, then so is X, and no updates to $\hat{H}$ are required. If X is a distinguished cluster but X' is not a distinguished cluster, then the update procedure to graph $\hat{H}$ is almost identical to that when a distinguished cluster becomes a non-distinguished cluster. Specifically, we apply a vertex-splitting operation to t, following which a supernode $v_{X'}$ representing the cluster X' is inserted into $\hat{H}$. The remaining details a straightforward and are omitted here.

Lastly, assume that a vertex x is deleted from some cluster $X \in \mathcal{X}$. In this case, we consider every edge $e \in E(H)$ that is incident to x one by one. Let y denote the other endpoint of e. If X is not a distinguished cluster, then we delete every copy of e that is incident to v_X from $\hat{H}$. Otherwise, we delete every copy of e that is incident to t from $\hat{H}$.

To summarize, we view a cluster-splitting operation that is applied to a cluster X as consisting of two steps. In the first step, a new cluster $X' \subseteq X$ is inserted into $\mathcal{X}$, and in the second step, the vertices of X' are deleted from X. Recall that, following such cluster-splitting updates, it is possible that for some edge $e \in E(H)$ whose endpoints lie in different clusters, the value $\text{skip}(e)$ increases. In such a case, we delete from $\hat{H}$ all copies of e, whose weight is below the new value $\text{skip}(e)$. This concludes the description of the algorithm for maintaining the graph $\hat{H}$.

It is easy to verify that the total time that is required in order to compute and maintain graph $\hat{H}$ is asymptotically bounded by the total time required to maintain graph H and the collection $\mathcal{X}$ of clusters times $O(\log^6 n)$. Indeed, from Claim 5.11, every cluster in $\mathcal{X}'_2$ has $O(\log n)$ close descendants, and so it may have at most $O(\log^2 n)$ descendant clusters overall. Therefore, every vertex of H may belong to $O(\log^2 n)$ clusters over the course of the algorithm. For each edge $e = (x, y)$, for every pair X, X' of clusters that contain x and y respectively, at any time during the algorithm, $O(\log^2 n)$ copies of e connecting the supernodes corresponding to X and X' may be present in $\hat{H}$. Overall, at most $O(\log^5 n)$ copies of an edge of H may ever be present in $\hat{H}$, and so the total time that is required in order to compute and maintain graph $\hat{H}$ is asymptotically bounded by the total time required to maintain graph H and the collection $\mathcal{X}$ of clusters times $O(\log^6 n)$.

Solving the DAG-like SSSP Problem on $\hat{H}$

So far we have defined a dynamic graph $\hat{H}$ with lengths and weights on edges and two special edges s and t, that undergoes an online sequence of edge-deletion and vertex-splitting updates, such that Property S1 holds. We consider an instance of the DAG-like SSSP problem defined on graph $\hat{H}$ with parameter $\Gamma = 64n \cdot 2^{8\sqrt{\log N}}$, parameter d replaced by $\frac{\Lambda}{4}$, and $\epsilon = \frac{1}{10}$. We apply the algorithm from Theorem 2.7 to the resulting instance of the DAG-like SSSP problem. Recall that the total update time of this algorithm is bounded by:

$$\tilde{O}\left((n')^2+|E(\hat{H})|+\Gamma\cdot n'\right)\leq \tilde{O}\left(n\cdot(n-|B^0|)\cdot 2^{8\sqrt{\log N}}\right)$$

Overall, the total time that is required for Part 1 of the algorithm, and also for maintaining graph $\hat{H}$ and for maintaining the data structures for the DAG-like SSSP problem from Theorem 2.7 is bounded by:

$$c_1\cdot n\cdot(n-|B^0|)\cdot 2^{c_2\sqrt{\log N}}\cdot(\log N)^{16c_2r+2c_2}.$$

Computing the Cut-Witness

Once the algorithm for the DAG-like SSSP problem terminates, we are guaranteed that graph $\hat{H}$ contains no s-t path Q with $\sum_{e\in E(Q)}\hat{\ell}(e)\leq\frac{\Lambda}{4}$ and $\sum_{e\in E(Q)}w(e)\leq\Gamma$.

We now define new lengths $\ell'(e)$ for edges $e\in E(H)$, as follows. For every regular edge $e\in E(H)$, we set $\ell'(e)=0$. Consider now a special edge $e\in E^{\mathsf{spec}}$. If both endpoints of e lie in B at the end of the algorithm, then we set $\ell(e)=0$. Otherwise, if $e\notin E^{\mathsf{bad}}$, then we set $\ell'(e)=\ell(e)$. Lastly, if $e\in E^{\mathsf{bad}}$, and not both endpoints of e lie in B, we set $\ell'(e)=\ell(e)+\operatorname{span}(e)\cdot\frac{\eta\cdot\log n}{8\Delta\cdot 2^{7\sqrt{\log N}}}$. We now show that $\{\ell'(e)\}_{e'\in E(H)}$ is a valid cut-witness.

Claim 5.15 *$\{\ell'(e)\}_{e'\in E(H)}$ is a valid cut-witness.*

Proof: Let $\tilde{E}$ denote all edges of G with both endpoints in B at the end of the algorithm. It is easy to verify that, for each edge $e\in\tilde{E}$, we have set $\ell'(e)=0$. Additionally, for every edge e that is incident to s or to t, we have set $\ell'(e)=0$. Next, we bound $\sum_{e\in E(H)}\ell'(e)$, as follows:

$$\begin{aligned}\sum_{e\in E(H)}\ell'(e)&\leq\sum_{e\in E^{\mathsf{spec}}\setminus\tilde{E}}\ell(e)+\sum_{e\in E^{\mathsf{bad}}\setminus\tilde{E}}\operatorname{span}(e)\cdot\frac{\eta\cdot\log n}{8\Delta\cdot 2^{7\sqrt{\log N}}}\\&\leq\sum_{e\in E^{\mathsf{spec}}\setminus\tilde{E}}\ell(e)+\frac{\eta\cdot\log n}{8\Delta\cdot 2^{7\sqrt{\log N}}}\cdot\sum_{e\in E^{\mathsf{bad}}}\operatorname{span}(e)\\&\leq\sum_{e\in E^{\mathsf{spec}}\setminus\tilde{E}}\ell(e)+\frac{(n-|B^0|)\log n}{2}\\&\leq\sum_{e\in E^{\mathsf{spec}}\setminus\tilde{E}}\ell(e)+\frac{\Lambda\cdot\Delta}{2\eta\log^4 n}.\end{aligned}$$

(we have used Inequality 9 and the fact that $\Lambda=(n-|B^0|)\cdot\frac{\eta\cdot\log^5 n}{\Delta}$).

Lastly, we prove that the distance from s to t in H, with respect to edge lengths $\ell'(\cdot)$, is at least $\frac{\Lambda}{32}$. Assume for contradiction that there is an s-t path Q in the current graph H, such that $\sum_{e\in E(Q)}\ell'(e)<\frac{\Lambda}{32}$. Recall that graph G contains no regular edge connecting a vertex of A to a vertex of B. Therefore, path Q contains at least one special edge. Moreover, if we discard the first and the last regular edges on Q, and denote by Q' the resulting path, then regular and special edges alternate on Q'. Since the length of every special edge in H is at least 1, we get that the number of regular edges on Q, that we denote by m_r, is bounded by $2\sum_{e\in E(Q)}\ell'(e)+3<\frac{\Lambda}{8}$.

Let $\hat{E}(Q) \subseteq E(Q)$ be the set of edges that also lie in the contracted graph $\hat{H}$, and denote $\hat{E}(Q) = (\hat{e}_1, \ldots, \hat{e}_z)$, where the edges are indexed in the order of their appearence on Q. For all $1 \le i \le z$, let $\hat{e}'_i$ be the copy of edge e_i in $\hat{H}$ that has the shortest length and the smallest weight. Then the sequence $(\hat{e}'_1, \ldots, \hat{e}'_z)$ of edges defines an s-t path in graph $\hat{H}$, that we denote by $\hat{Q}$. Notice that $\hat{E}(Q)$ may not contain an edge with both endpoints in the current set B. Therefore, for every special edge $e'_i \in \hat{E}(Q)$, $\hat{\ell}(\hat{e}'_i) = \ell(\hat{e}_i) \le \ell'(\hat{e}_i)$, while for every regular edge $e \in \hat{E}(Q)$, $\hat{\ell}(e) = 1$. We conclude that the length of path $\hat{Q}$ in graph $\hat{H}$ is bounded by $\sum_{e \in E(Q)} \ell'(e) + m_r \le \frac{\Lambda}{4}$. To reach a contradiction, it is now enough to show that $\sum_{\hat{e}'_i \in \hat{E}(Q)} w(\hat{e}'_i) \le \Gamma$.

Indeed, recall that, for every edge $e \in E^{\mathsf{bad}}$, we have set $\ell'(e) = \ell(e) + \operatorname{span}(e) \cdot \frac{\eta \cdot \log n}{8\Delta \cdot 2^{7\sqrt{\log N}}}$. Since $\sum_{e \in E(Q)} \ell'(e) \le \frac{\Lambda}{32}$, we get that:

$$\sum_{e \in E(Q) \cap E^{\mathsf{bad}}} \operatorname{span}(e) \cdot \frac{\eta \cdot \log n}{8\Delta \cdot 2^{7\sqrt{\log N}}} \le \frac{\Lambda}{32}.$$

Therefore:

$$\sum_{e \in E(Q) \cap E^{\mathsf{bad}}} \operatorname{span}(e) \le \frac{\Lambda \cdot \Delta \cdot 2^{7\sqrt{\log N}}}{4\eta \log n} = \frac{(n - |B^0|) \cdot 2^{7\sqrt{\log N}} \cdot \log^4 n}{4},$$

since $\Lambda = (n - |B^0|) \cdot \frac{\eta \cdot \log^5 n}{\Delta}$. But then:

$$\sum_{e \in E(Q)} \operatorname{skip}(e) \le n + 2 \sum_{e \in E(Q) \cap E^{\mathsf{bad}}} \operatorname{span}(e) \le n + \frac{(n - |B^0|) \cdot 2^{7\sqrt{\log N}} \cdot \log^4 n}{2} \le \frac{\Gamma}{4},$$

since $\Gamma = 64n \cdot 2^{8\sqrt{\log N}}$. Since, for all $1 \le i < z$, $w(\hat{e}'_i) \le 2\operatorname{skip}(\hat{e}_i)$, we get that:

$$\sum_{\hat{e}'_i \in E(\hat{Q})} w(\hat{e}'_i) \le 2 \sum_{e \in E(Q)} \operatorname{skip}(e) \le \frac{\Gamma}{2},$$

a contradiction. □

5.3 Completing the Algorithm

Throughout the algorithm, for every special edge $e = (x, y)$ in graph H, we maintain a counter $n(e)$, that counts the number of s-t paths containing e that the algorithm reported in response to queries that contain e, since $\ell(e)$ was doubled last, or since the beginning of the algorithm – whatever happened later.

The algorithm contains at most Δ iterations, and terminates when either (i) all iterations are completed; or (ii) the algorithm for the DAG-like SSSP problem returns "FAIL"; or (iii) one of the algorithms for the MaintainCluster problem or for the MaintainSpecialCluster problem that we use to maintain the clusters of $\mathcal{X}$ returns "FAIL". In the latter case, we terminate our algorithm with a "FAIL"; we have shown that this may only happen with probability at most 1/2. In the second case, we return the cut-witness as described above.

We now proceed to describe the ith iteration. We use the algorithm for the DAG-like SSSP problem from Theorem 2.7 to compute a simple s-t path P_i in the current graph $\hat{H}$, with $\sum_{e \in E(\hat{H})} \hat{\ell}(e) \le$

$(1+10\epsilon)\cdot\frac{\Lambda}{4}\leq\frac{\Lambda}{2}$, in time $O(|E(P_i)|)$. We denote by $\hat{e}_1,\hat{e}_2,\ldots,\hat{e}_z$ the sequence of the edges on this path. For all $1\leq j\leq z$, let $e_j=(x_j,y_j)$ be the edge corresponding to $\hat{e}_j$ in graph H. Notice that $x_1=s$, and, for all $1\leq j<z$, there is a cluster of $\mathcal{X}$ that contains both y_j and x_{j+1}; we denote this cluster by X_j. Cluster X_j cannot be a distinguished cluster, since path P_i is simple. Lastly, there must be a distinguished cluster $X_z\in\mathcal{X}$, that contains y_z.

Recall that, if Case 1 happened, then $|U^0|\geq 2(n-|B^0|)\cdot(\log N)^{128}$. Moreover, $|U|\geq\gamma$ holds throughout the algorithm, where $\gamma=|U^0|-256(n-|B^0|)(\log N)^{64}\geq(n-|B^0|)\cdot(\log N)^{128}$. At the end of the algorithm, $|V(G)\setminus B|\leq(n-|B^0|)+\Delta\leq 2(n-|B^0|)$ holds. Therefore, if Case 1 happens, then U remains a distinguished cluster throughout the algorithm.

For all $1\leq j<z$, we query Algorithm $\mathcal{A}(X_j)$ for the $\mathsf{MaintainCluster}$ problem with vertices y_j and x_{j+1}. Let $Q_j\subseteq H[X_j]$ be the simple path connecting y_j to x_{j+1} that the algorithm returns, whose length is at most d_{X_j}. Assume first that either Case 1 did not happen; or Case 1 happened and $X_z\neq U$. In this case, we select an arbitrary vertex $x_{z+1}\in B\cap X_z$, and query Algorithm $\mathcal{A}(X_j)$ for the $\mathsf{MaintainCluster}$ problem with vertices y_z and x_{z+1}. Let $Q_z\subseteq H[X_z]$ be the simple path connecting y_z to x_{z+1} that the algorithm returns, whose length is at most d_{X_z}. If Q_z contains any vertex of B as an inner vertex, then we truncate it, so it connects y_z to a vertex of B, and does not contain any vertex of B as an inner vertex. We then append an edge connecting the last vertex of Q_z to t to the path Q_z. (For all $1\leq j\leq z$, if X_j is the singleton cluster, then we simply let Q_j contain a single vertex $x_j=y_{j+1}$). If Case 1 happened and $X_z=U$, then we query Algorithm $\mathcal{A}'(U)$ with vertex y_z, to obtain a simple path Q_z connecting y_z to some vertex $b\in B$, such that Q_z has length at most d_U and no inner vertices of Q_z belong to B. We then append the edge (b,t) at the end of path Q_z.

By concatenating $e_1,Q_1,e_2\ldots,e_z,Q_z$, we obtain an s-t path in the current graph H, that we denote by P'_i. Since the initial path P_i was simple, all custers in $\mathcal{X}$ are mutually disjoint, and each of the paths $Q_1,\ldots,Q_z$ is simple, we get that P'_i is a simple path as well. We now bound the length of P'_i. Recall that $\sum_{j=1}^{z}\ell(e_j)\leq\sum_{j=1}^{z}\hat{\ell}(\hat{e}_j)\leq\frac{\Lambda}{2}$.

Recall that, for every cluster $X\in\tilde{\mathcal{X}}'_2$, $d_X=\min\left\{\frac{|X^0|\cdot\eta}{\Delta},d\right\}$. Notice that clusters $X_1,\ldots,X_{z-1}$ may not contain vertices of B. Since, at the end of the algorithm, $|B|\geq|B^0|-\Delta$, we get that:

$$\sum_{j=1}^{z-1}|X_j|\leq n-|B|\leq n-|B^0|+\Delta\leq 2(n-|B^0|).$$

Moreover, for every cluster $X\in\tilde{\mathcal{X}}'_2$, algorithm $\mathcal{A}(X)$ terminates once $|X|$ falls below $|X^0|/2$, and so:

$$\sum_{j=1}^{z-1}|X_j^0|\leq 2\sum_{j=1}^{z-1}|X_j|\leq 4(n-|B^0|).$$

Altogether, we get that:

$$\sum_{j=1}^{z-1}d_{X_j}\leq\sum_{j=1}^{z-1}\frac{|X_j^0|\cdot\eta}{\Delta}\leq\frac{4(n-|B^0|)\cdot\eta}{\Delta}\leq\frac{\Lambda}{4},$$

since $\Lambda=(n-|B^0|)\cdot\frac{\eta\cdot\log^5 n}{\Delta}$.

Note also that either $d_{X_z}\leq d_U$ (if X_z is the special cluster), or $d_{X_z}\leq d$ (otherwise) must hold. Recall that $d_U=\max\left\{(\log N')^{64},\left(|U^0\setminus B^0|+\Delta\right)\frac{\eta}{4\Delta\cdot 2^{\sqrt{\log N}}}\right\}$. Since $\Lambda=(n-|B^0|)\cdot\frac{\eta\cdot\log^5 n}{\Delta}$, while

$|U^0 \setminus B^0| + \Delta \le 2(n - |B^0|)$ and $\frac{(n-|B^0|)\cdot\eta}{\Delta} > 2^{4\sqrt{\log N}}$ from Inequality 2, we get that $d_U \le \frac{\Lambda}{4}$. Recall also that $d = \frac{(n-|B^0|)\cdot\eta}{4\Delta\cdot 2^{\sqrt{\log N}}} \le \frac{\Lambda}{4}$. Therefore, $d_z \le \frac{\Lambda}{4}$ must hold. Overall, we get that:

$$\sum_{e\in E(P')} \ell(e) \le \sum_{j=1}^{z} \ell(e_j) + \sum_{j=1}^{z-1} d_{X_j} \le d_{X_z} \le \Lambda,$$

as required. We increase the counter $n(e)$ for every edge $e \in E(P')$. If, for any such edge e, the counter $n(e)$ reaches η, then we double the length of e, and we reset $n(e) = 0$.

Notice that the time required to respond to the queries posed to algorithms $\mathcal{A}(X_1), \ldots, \mathcal{A}(X_{z-1})$, and Algorithm $\mathcal{A}(X_z)$ (if X_z is a regular cluster) or Algorithm $\mathcal{A}'(X_z)$ (if it is a special cluster) is already accounted for in the analysis of the running times of these algorithms. The additional time that is required in order to respond to the query is bounded by $O(|E(P_i)|)$.

Overall, the additional time that is required in order to respond to all queries is bounded by $\sum_{i=1}^{\Delta} O(|E(P_i)|) \le \sum_{i=1}^{\Delta} O(|E(P'_i)|)$. Recall that, for every special edge $e \in E(H)$, the length of e in H is doubled once it participates in η paths that the algorithm returns. Once the length of e reaches Λ, it may no longer participate in any path that the algorithm returns. Therefore, every special edge may participate in at most $O(\eta \cdot \log \Lambda) \le O(\eta \log N)$ paths $P'_1, \ldots, P'_\Delta$. Since there are at most n special edges in H, and since regular and special edges alternate on the paths $P'_1, \ldots, P'_\Delta$, we get that $\sum_{i=1}^{\Delta} |E(P'_i)| \le O(n\eta \log N) \le O(n\Delta \log N) \le O(n \cdot (n - |B^0|) \cdot \log N)$, since $\eta \le \Delta$ and $\Delta \le n - |B^0|$ from the problem definition. The total additional time that is required in order to respond to all queries is then bounded by $O(n \cdot (n - |B^0|) \cdot \log N)$, and the total running time of the algorithm is bounded by:

$$O\left(c_1 \cdot n \cdot (n - |B^0|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r + 2c_2}\right) \le c_1 \cdot n \cdot (n - |B^0|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r + 7c_2}.$$

6 Layered Connectivity Data Structure and an Algorithm for the MaintainSpecialCluster problem

The goal of this section is to prove the second part of Theorem 4.12, by showing that an algorithm for the r-restricted RouteAndCut problem implies an algorithm for the r-restricted MaintainSpecialCluster problem with the required running time. In order to prove this theorem, we define a new data structure, that we call a Layered Connectivity Data Structure, and provide an algorithm for maintaining this data structure in a graph that undergoes online increases in the lengths of its edges. The same data structure, and the same algorithm for maintaining it, will also be used in Section 7 in order to prove the first part of Theorem 4.12, namely that an algorithm for the r-restricted RouteAndCut problem implies an algorithm for the r-restricted MaintainCluster problem with the required running time. Because of this, we define both the Layered Connectivity Data Structure, and the algorithm for maintaining this data structure slightly more generally than what is needed for the MaintainSpecialCluster problem. But first we start with intuition.

Recall the definition of the MaintainSpecialCluster problem. At a high level, the input is a perfect well-structured graph $G = (L, R, E)$ with a proper assignment of lengths $\ell(e)$ on its edges $e \in E$ that are given in the adjacency list representation, together with a subset $\beta \subseteq V(G)$ of vertices with $|\beta| \ge \frac{99|V(G)|}{100}$, and parameters $N \ge W^0(G)$, $1 \le \Delta \le \frac{|\beta|}{(\log N)^{128}}$, $1 \le \eta \le \Delta$, and $d \ge (\log N)^{64}$, such that $\frac{\Delta \cdot d}{\eta} \le |V(G)| - |\beta| + \Delta \cdot (\log N)^{64}$, and every special edge has length 1. Over the course of the algorithm, vertices may be deleted from set β, and the lengths of some edges may be increased, but the length of every special edge incident to a vertex that currently lies in β remains 1.

The algorithm consists of at most Δ iterations. At the beginning of every iteration i, the algorithm is given a vertex $x \in V(G)$, and it must return a simple path P_i in G connecting x to any vertex $y \in \beta$, such that the length of the path is at most d, and no inner vertices on the path lie in β. After that, the length of some special edges on path P_i may be doubled, and vertex y is deleted from β. Recall that, for any edge $e \in E(G)$, if $\ell(e)$ is doubled at times τ' and τ with $\tau' < \tau$, then at least η paths that the algorithm returns during the time interval $[\tau', \tau)$ must contain e. The algorithm may, at any time, produce a strongly well-structured cut (X, Y) in G of sparsity $\Phi_G(X, Y) \leq \frac{(\log N)^{64}}{d}$, such that $w(X) \geq 1.8|\beta \cap X|$ holds. Let $J' \subseteq Y$ denote the set of vertices that serve as endpoints of the edges of $E_G(X, Y)$. The vertices of $X \cup J'$ are then deleted from G, and the algorithm continues with the resulting graph $G = G[Y \setminus J']$.

A natural way to solve this problem is to simply maintain a collection $\mathcal{Q} = \{K(v) \mid v \in V(G)\}$ of paths in G, where for each vertex $v \in V(G)$, path $K(v)$ connects v to some vertex of β, such that the length of every path in $\mathcal{Q}$ is at most d, and every vertex of β serves as endpoint of a small number of such paths. Then, whenever a query vertex x arrives, we could simply return the path $K(x)$ in response to the query, possibly truncating it to ensure that no inner vertices on the path belong to β. As the time progresses, some of the paths in $\mathcal{Q}$ may however be destroyed. For example, as the lengths of some special edges are doubled, the lengths of some paths in $\mathcal{Q}$ may become too large, so we can no longer use them in order to respond to queries. Additionally, some vertices may be deleted from β, and so paths connecting vertices of G to such vertices can no longer be used in order to respond to queries. Lastly, when our algorithm returns a sparse cut (A, B), and some of the vertices are deleted from G following this, it is possible that, for some of the paths $K(v) \in \mathcal{Q}$, some inner vertices on $K(v)$ are deleted from G, while the first endpoint of the path remain in G. Whenever, for some vertex $v \in V(G)$, the corresponding path $K(v) \in \mathcal{Q}$ is destroyed, we say that vertex v became *disconnected*. We need to ensure that, before we respond to each query, every vertex of G has a valid path $K(v) \in \mathcal{Q}$. Therefore, whenever a vertex v becomes disconnected, we need to compute a new path $K(v)$ connecting it to some vertex that currently lies in β. The difficulty is that, even if the number of vertices that are currently disconnected is quite small, if we attempt to connect them to the vertices of β directly, the amount of time that we need to invest to compute such a routing may be potentially as large as $\Omega(|V(G)| \cdot (|V(G)| - |\beta|)$. We note that we can ensure that the total number of vertices that ever become disconnected over the course of the algorithm (counting every time a vertex becomes disconnected separately, even if the same vertex is disconnected several times), is relatively small. However, there may be a large number of iterations, in each of which a small number of vertices become disconnected. Therefore, computing a new collection of paths connecting these disconnected vertices to β from scratch in every iteration is prohibitively expensive in terms of the running time. A different approach that could address this issue is the following: let U be the set of vertices that are currently disconnected, and let $U' = V(G) \setminus U$. Instead of connecting the vertices of U to the vertices of β directly, we can instead try to connect them to the vertices of U', and then exploit the paths in $\{K(v) \mid v \in U'\}$ in order to route the vertices of U to the vertices of β. In doing so we will only need to explore a relatively small subgraph of G, and we can use the algorithm for the r-restricted RouteAndCut problem, whose existence is assumed in the statement of Theorem 4.12, in order to do so in time roughly $O(W(G)^{1+o(1)} \cdot |U|)$. The difficulty with this approach is that, as we compose the routing paths with each other, iteration after iteration, we may obtain routing paths that are very long. To conclude, we need to balance between the need to ensure that all paths in $\mathcal{Q}$ are sufficiently short, with the need to ensure that the total running time invested in reconnecting the disconnected vertices to β is not too high. We use a natural *layering* approach, by defining a Layered Connectivity Data Structure, which we describe below.

So far we provided intuition for the Layered Connectivity Data Structure in the context of obtaining an algorithm for the r-restricted MaintainSpecialCluster problem. We will also exploit this data structure in our algorithm for the r-restricted MaintainCluster problem. Intuitively, instead of the set β of vertices that is given as part of input to the MaintainSpecialCluster problem, in order to solve the MaintainCluster problem, we will select a random set β of such vertices of an appropriate cardinality, and then embed an expander that is defined over the vertices of β into the input graph G. The Layered Connectivity Data Structure data structure will then be used to maintain paths connecting every vertex of G to some vertex of β. We will also maintain a similar data structure in graph $\overleftarrow{G}$, that is obtained by reversing the direction of every edge in G, obtaining a collection of paths that, for every vertex $v \in V(G)$, connects some vertex of β to v. We now formally define the Layered Connectivity Data Structure data structure, followed by an algorithm for maintaining such a data structure in dynamic graphs that arise from the MaintainCluster and MaintainSpecialCluster problems. Lastly, we design an algorithm for the MaintainSpecialCluster problem that relies on this data structure.

6.1 The Layered Connectivity Data Structure

We assume that we are given a perfect well-structured graph $G = (L, R, E)$ with $|V(G)| = n$, and a proper assignment $\ell(e)$ of lengths to the edges $e \in E$, that are given in the adjacency-list representation. Recall that for every vertex $v \in V(G)$, its weight $w(v)$ is the length of the unique special edge that is incident to v in G. We denote $W = \sum_{v\in V(G)} w(v)$, and we let H be the subdivided graph corresponding to G (see Definition 4.1). We assume further that we are given a parameter $N \geq \max\{n, W/32\}$ that is greater than a large enough constant, together with additional parameters $d' \geq 1$, $\rho \geq \log^2 N$, and $1 \leq r \leq \lceil\sqrt{\log N}\rceil$, such that $d' \leq \min\left\{\frac{2^{r\sqrt{\log N}}}{\log^8 N}, \frac{n}{8}\right\}$. Lastly, we are given a collection $\beta \subseteq V(H)$ of vertices, that we refer to as *centers*, with $|\beta| \geq \frac{W}{d'\cdot \log^2 N}$.

Let $\lambda = \lceil\log(64N)\rceil + 1$. A Layered Connectivity Data Structure for graph G consists of the following ingredients:

- The first ingredient is a partition $(U_0, U_1, \ldots, U_\lambda)$ of $V(H)$ into disjoint subsets, such that $U_0 = \beta$. For all $0 \leq i \leq \lambda$, we refer to the vertices in set U_i as *layer-i vertices*, and we sometimes refer to the vertices of U_λ as *disconnected vertices*. We also denote by $U^{\geq i} = \bigcup_{i'=i}^{\lambda} U_{i'}$ and by $U^{\leq i} = \bigcup_{i'=0}^{i} U_{i'}$. Similarly, we denote by $U^{>i} = \bigcup_{i'=i+1}^{\lambda} U_{i'}$ and by $U^{<i} = \bigcup_{i'=0}^{i-1} U_{i'}$.
- The second ingredient is collections $\mathcal{Q}_1, \ldots, \mathcal{Q}_{\lambda-1}$ of paths in graph H. For all $1 \leq i \leq \lambda - 1$, $\mathcal{Q}_i = \{K(v) \mid v \in U_i\}$, where, for every vertex $v \in U_i$, path $K(v)$ connects vertex v to some vertex $v' \in U^{<i}$. We say that vertex v' is the *parent-vertex* of v, and vertex v is a *child-vertex* of v'. We also say that v is a *descendant* of v'. The descendant relation is transitive: if v is a descendant of v', and v' is a descendant of v'', then v is a descendant of v''. Additionally, every vertex is a descendant of itself. If vertex x is a descendant of vertex y, then we say that y is an *ancestor* of x. Note that ancestor-descendant relation is only determined by the endpoints of the paths in $\bigcup_{i=1}^{\lambda-1} \mathcal{Q}_i$, and it is independent of the inner vertices on these paths. For every vertex $x \in \beta$ and index $0 \leq i < \lambda$, we denote by $\mathcal{D}_i(x)$ the set of all vertices $v \in U_i$ that are descendants of x, and we denote by $\mathcal{D}(x) = \bigcup_{i=0}^{\lambda-1} \mathcal{D}_i(x)$ the set of all descendant-vertices of x.

  Lastly, we require that the following properties hold for a Layered Connectivity Data Structure:

L1. For all $1 \leq i < \lambda$, the paths in $\mathcal{Q}_i$ have length at most $4d' \log^{10} N$ each, and they cause edge-congestion at most $4d' \log^9 N$ in H;

L2. For every vertex $x \in \beta$, for every layer $1 \leq i \leq \lambda - 1$, $|\mathcal{D}_i(x)| \leq \rho$; and

L3. For every layer $1 \leq i \leq \lambda - 1$, every vertex $v \in U^{<i} \setminus U^0$, may serve as an endpoint of at most one path in $\mathcal{Q}_i$.

The Layered Connectivity Data Structure data structure must also maintain, for every vertex $x \in \beta$ and layer $1 \leq i < \lambda$, the collection $\mathcal{D}_i(x) \subseteq U_i$ of layer-i descendants of x. Additionally, for every layer $1 \leq i \leq \lambda - 1$ and vertex $v \in U^{<i}$, if v serves as an endpoint of a path $K(u) \in \mathcal{Q}_i$, then we maintain a pointer from v to u.

In order to construct and maintain a Layered Connectivity Data Structure, as graph G undergoes updates, we will employ a procedure that we call ReconnectLayer, and describe next.

6.2 Procedure ReconnectLayer

As before, we assume that we are given a perfect well-structured graph $G = (L, R, E)$ with $|V(G)| = n$, and a proper assignment $\ell(e)$ of lengths to the edges $e \in E$, that are given in the adjacency-list representation, together with a parameter $N \geq \max\{n, W/32\}$ that is greater than a large enough constant, and parameters $d' \geq 1$, $\rho \geq \log^2 N$, and $1 \leq r \leq \lceil\sqrt{\log N}\rceil$, where $W = \sum_{v \in V(G)} w(v)$, and $d' \leq \min\left\{\frac{2^{r\sqrt{\log N}}}{\log^8 N}, \frac{n}{8}\right\}$. We additionally assume that $\frac{W \cdot \log^3 N}{|\beta|} \leq \frac{\rho}{16}$, and that the length of every edge in G is at most d'. Lastly, we assume that we are given a subdivided graph H corresponding to G, in the adjacency-list representation, and a subset $\beta \subseteq V(H)$ of its vertices, with $|\beta| \geq \frac{W}{d' \cdot \log^2 N}$.

A valid input to Procedure ReconnectLayer consists of an index $1 \leq i < \lambda$ of a layer, and a valid Layered Connectivity Data Structure for G, such that, for all $i \leq i' < \lambda$, $U_{i'} = \emptyset$, and $|U_\lambda| \leq 2^{\lambda - i}$. We denote $k = |U_\lambda|$. A valid output of Procedure ReconnectLayer is one of the following:

- either a valid Layered Connectivity Data Structure, in which, for all $i < i' < \lambda$, $U_{i'} = \emptyset$ and $|U_\lambda| \leq \frac{k}{4\lambda}$ holds; additionally, layers $U_1, \ldots, U_{i-1}$, together with the corresponding collections of paths $\mathcal{Q}_1, \ldots, \mathcal{Q}_{i-1}$ must remain the same as in the input Layered Connectivity Data Structure; or
- a strongly well-structured $\frac{1}{4d'}$-sparse cut (A, B) in G with $w(A), w(B) \geq \frac{k}{2^{14} \log N}$ and $w(A) \geq 1.8|\beta \cap A|$; or
- A weakly well-structured cut (A', B') in H with $|A'| \geq \frac{511|V(H)|}{512}$, $|A' \cap \beta| < \frac{2|\beta|}{\log^3 N}$, and $|E_H(A', B')| \leq \frac{|V(H)|}{1024 d' \cdot \log^6 N}$.

Next, we show an algorithm, that, given a valid input to Procedure ReconnectLayer, produces a valid output for the procedure. The algorithm relies on Algorithm $\mathcal{A}$ for the r-restricted RouteAndCut problem, whose existence was assumed in the statement of Theorem 4.12.

Claim 6.1 *Suppose that, for some parameter $r \geq 1$, there exists a randomized algorithm for the r-restricted* RouteAndCut *problem, that, given as an input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most $c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2}$. Then there is a randomized algorithm, that, given a valid input to Procedure* ReconnectLayer*, where graph G and its corresponding subdivided graph H are given in the adjacency-list representation, with the same parameter r, either produces a valid output for the procedure, or returns "FAIL". The running time of the algorithm is at most $O\left(c_1 \cdot W \cdot k \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+14}\right)$, and the probability that it returns "FAIL" is at most $\frac{1}{N^{10}}$.*

The remainder of this subsection is dedicated to the proof of Claim 6.1. We denote by $\mathcal{A}$ the algorithm for the r-restricted $\mathsf{RouteAndCut}$ problem, whose existence is assumed in the statement of the claim.

The most natural way to design an algorithm for Procedure $\mathsf{ReconnectLayer}$ is to simply cast it as a special case of the r-restricted $\mathsf{RouteAndCut}$ problem, in the underlying graph H, with the set $A = U_\lambda$ of source vertices, and the set $B = \bigcup_{i'=0}^{i-1} U_{i'}$ of destination vertices. For technical reasons, we must allow every vertex $x \in \beta$ to serve as an endpoint of at least $\Theta\left(\frac{W \cdot \text{poly}\log N}{|\beta|}\right)$ of the resulting routing paths, while each of the remaining vertices of B may serve as an endpoint of at most one path. To accommodate this requirement, we slightly modify graph H. The modification essentially replaces every vertex $x \in \beta$ with $\Theta\left(\frac{W \cdot \text{poly}\log N}{|\beta|}\right)$ distinct vertices, while ensuring that the resulting graph remains a uniform well-structured graph. The main difficulty with this approach, however, is that, for every vertex $x \in \beta$, and every layer $1 \leq i' < i$, there may be up to ρ vertices of $U_{i'}$ that are descendants of x. It is possible that we will connect a distinct vertex of U_λ to each such descendant vertex of x via the routing computed by the algorithm for the $\mathsf{RouteAndCut}$ problem. The vertices of U_λ from which these routing paths originate are then added to layer U_i, and, as the result, the number of descendants of x in U_i may become too high. In order to overcome this difficulty, we will iteratively solve 4λ instances of the $\mathsf{RouteAndCut}$ problem, with the goal of constructing, for every vertex $v \in U_\lambda$, a collection $\mathcal{R}(v)$ of at most 4λ paths that originate at v. Let $U' \subseteq U_\lambda$ be the subset of vertices v with $|\mathcal{R}(v)| \geq 2\lambda$, and let $\mathcal{R} = \bigcup_{v \in U'} \mathcal{R}(v)$. Our goal is to ensure that set U' contains almost all vertices of U_λ, the paths in $\mathcal{R}$ cause low edge-congestion, and they terminate at the vertices of $\bigcup_{i'=0}^{i-1} U_{i'}$. We also need to ensure that every vertex of β serves as an endpoint to at most $\Theta\left(\frac{W \cdot \text{poly}\log N}{|\beta|}\right)$ paths in $\mathcal{R}$, while every vertex of $\bigcup_{i'=1}^{i-1} U_{i'}$ serves as an endpoint of at most one such path. Intuitively, by sending $\frac{1}{2\lambda}$ flow units on every path in $\mathcal{R}$, we obtain a fractional flow where every vertex in U' sends at least 1 flow unit to the vertices of $\bigcup_{i'=0}^{i-1} U_{i'}$, and, for every vertex $x \in \beta$, the descendants of x, including x itself, receive at most ρ flow units. We can then compute an integral flow, which defines, for every vertex $v \in U'$, a path $K(v)$ connecting v to some vertex in $\bigcup_{i'=0}^{i-1} U_{i'}$, such that, for every vertex $x \in \beta$, at most ρ such paths terminate at the descendant vertices of x.

We now provide a formal description of the procedure. Recall that we have denoted $k = |U_\lambda|$. We assume first that $k \geq 2^{20} \cdot d' \cdot \log^6 N$. We show an algoritm for the special case where $k < 2^{20} \cdot d' \cdot \log^6 N$ later. We start by describing the modified graph H', to which the algorithm for the $\mathsf{RouteAndCut}$ problem will be applied iteratively. Throughout, we use a parameter $M = \frac{W \cdot \log^3 N}{|\beta|}$. Since $\frac{W \cdot \log^3 N}{|\beta|} \leq \frac{\rho}{16}$ from the problem definition, we get that $M \leq \frac{\rho}{16}$.

6.2.1 Modified Graph H'.

Recall that we are given as input a perfect well-structured graph G, and its corresponding subdivided graph H. Both graphs are given in the adjacency-list representation. We first define the graph H', and then describe an algorithm to compute its adjacency-list representation efficiently.

Graph H' is defined as follows. We start with $H' = H$, and the bipartition (L'', R'') of its vertices, that is identical to the bipartition (L', R') of $V(H)$. We then inspect every vertex $x \in \beta$ one by one. If $x \in L''$, then we add the collection $\{x_1, \ldots, x_{M-1}\}$ of new vertices to graph H', that we call *copies* of x. The vertices in $\{x_1, \ldots, x_{M-1}\}$ are also added to R'', and, for all $1 \leq j \leq M-1$, we add a regular edge (x, x_j) to graph H'. If $x \in R''$, then we also add a collection $\{x_1, \ldots, x_{M-1}\}$ of new vertices to H', called copies of x, but now we add x_1 to L'', and $x_2, \ldots, x_{M'-1}$ to R''. We add a special edge (x, x_1), and, for all $2 \leq j \leq M-1$, we add a regular edge (x_1, x_j) to H'. This concludes the definition of graph H'.

Note that H' is a uniform well-structured graph. Since $M = \frac{W \cdot \log^3 N}{|\beta|}$, and since $|V(H)| = W$, we get that $|V(H')| \leq O(W \log^3 N)$, and moreover, $|E(H') \setminus E(H)| \leq O(W \log^3 N)$. It is easy to modify the adjacency-list representation of H to obtain the adjacency-list representation of H', by inserting all vertices of $V(H') \setminus V(H)$ and all edges $E(H') \setminus E(H)$ into the former, in time $O(W \log^3 N)$.

6.2.2 Parameters for the RouteAndCut Problem Instances

Our algorithm will perform 4λ iterations. In every iteration, we will apply Algorithm $\mathcal{A}$ for the r-restricted RouteAndCut problem to graph H', with suitably chosen sets A' and B' of vertices of H'. We now define the parameters that will be used in each such instance of the RouteAndCut problem, and verify that this setting of the parameter indeed defines a valid instance of the problem.

We start by letting $N' = 64N \log^3 N$. It is easy to see that:

$$\log N' \leq \log N + 6 \log\log N \leq \left(1 + \frac{1}{64c_2 r}\right) \log N, \tag{10}$$

since $r \leq \left\lceil \sqrt{\log N} \right\rceil$ and N is large enough. Additionally, $\sqrt{\log N'} \leq \sqrt{\log N + 6 \log\log N} \leq \sqrt{\log N} + \sqrt{6 \log\log N}$. Therefore:

$$2^{\sqrt{\log N'}} \leq 2^{\sqrt{\log N}} \cdot \log^6 N \leq 2^{2\sqrt{\log N}}. \tag{11}$$

We will use these facts later.

We also use parameters $\Delta' = k$, and $\eta' = 2^{20} \cdot d' \cdot \log^6 N$. In all instances of the RouteAndCut problem that we construct, we will ensure that $U_\lambda \subseteq A'$, while set B' contains all vertices of U_0 and their copies, and additionally, $|V(H') \setminus B'| \leq 4\lambda k$ holds. We show that all such resulting instances are valid instances of the r-restricted RouteAndCut problem in the following simple claim, whose proof is deferred to Section C.1 of Appendix.

Claim 6.2 *Let A', B' be any pair of disjoint subsets of $V(H')$ with $U_\lambda \subseteq A'$, and with all vertices of U_0 and their copies lying in B', such that $|V(H') \setminus B'| \leq 4\lambda k$. Then $(H', A', B', \Delta', \eta', N')$ is a valid instance of the r-restricted* RouteAndCut *problem.*

We are now ready to describe our algorithm, that consists of at most 4λ iterations. The algorithm starts with a collection $\mathcal{R} = \emptyset$ of paths, and in every iteration, at most k paths are added to $\mathcal{R}$. We now describe a single iteration.

6.2.3 Description of an Iteration

We describe the jth iteration, for $1 \leq j \leq 4\lambda$. We construct a set B_j of vertices of H' as follows. We add to B_j all vertices of β and their copies. Additionally, we add to B_j all vertices of $U^{<i} \setminus U_0$, except for those that serve as endpoints of the paths that currently lie in $\mathcal{R}$. Notice that set $V(H') \setminus B_j$ contains all vertices of U_λ, and, additionally, every vertex $v \in U^{<i} \setminus U_0$ that serves as an endpoint of some path that currently belongs to $\mathcal{R}$. Since at most k paths are added to $\mathcal{R}$ in every iteration, and since the number of iterations is bounded by 4λ, we get that $|V(H') \setminus B_j| \leq 4\lambda \cdot k$. We also set $A_j = U_\lambda$, and we consider the resulting instance $(H', A_j, B_j, \Delta', \eta', N')$ of the RouteAndCut problem. From Claim 6.2, this is a valid r-restricted instance of the problem, and therefore we can apply Algorithm $\mathcal{A}$ to this instance. Recall that Algorithm $\mathcal{A}$ may return "FAIL", with probability at most

1/2. In this case, we simply repeat Algorithm $\mathcal{A}$, until either it does not return "FAIL", or until it returns "FAIL" in at least $\log^3 N$ consecutive iterations. In the latter case, we terminate our algorithm and return "FAIL", and we say that iteration j of the algorithm terminated with a " FAIL". Notice that the probability that iteration j terminates with a "FAIL" is at most $2^{-\log^3 N}$. We now assume that iteration j did not terminate with a "FAIL", and we consider the output of the last execution of Algorithm $\mathcal{A}$. Let $\mathcal{R}'_j$ be the routing that the algorithm returned. We consider two cases, depending on whether $|\mathcal{R}'_j| \geq \left(1 - \frac{1}{64\lambda}\right) k$ or not.

Case 1: $|\mathcal{R}'_j| \geq \left(1 - \frac{1}{64\lambda}\right) k$

Recall that the paths in $\mathcal{R}'_j$ cause congestion at most $4\eta' \cdot \log N'$. We can assume w.l.o.g. that, for every path $R \in \mathcal{R}'_j$, if $x \in B_j$ is an inner vertex on path R, then x also serves as an endpoint of another path in $\mathcal{R}'_j$ (as otherwise we can truncate path R, so that it terminates at x).

We say that a path $Q \in \mathcal{R}'_j$ is *long* if Q contains at least $d' \cdot \log^{10} N$ special edges, and we say that it is *short* otherwise. We use the next simple claim in order to show that $\mathcal{R}'_j$ contains many short paths.

Claim 6.3 *The number of long paths in* $\mathcal{R}'_j$ *is at most* $\frac{k}{64\lambda}$.

Proof: Let E' be the set of all special edges of H' that participate in the paths in $\mathcal{R}'_j$. We claim that $|E'| \leq 6\lambda k$. Indeed, we can partition E' into two subsets: set E'_1 containing all special edges e, such that at least one endpoint of e lies in $V(H') \setminus B_j$, and set E'_2 containing all remaining edges. Since $|V(H') \setminus B_j| \leq 4\lambda k$, we get that $|E'_1| \leq 4\lambda k$. From the above discussion, for every edge $e \in E'_2$, at least one endpoint of e must serve as an endpoint of a path in $\mathcal{R}'_j$, and so $|E'_2| \leq |\mathcal{R}'_j| \leq k$. Therefore, overall, $|E'| \leq |E'_1| + |E'_2| \leq 6\lambda k$. Since the paths in $\mathcal{R}'_j$ cause congestion at most $4\eta' \log N'$, we get that: $\sum_{Q \in \mathcal{R}'_j} |E(Q) \cap E'| \leq |E'| \cdot 4\eta' \cdot \log N' \leq 24\eta'\lambda k \log N' \leq 512 k\eta' \log^2 N$, since $\lambda = \lceil \log(64N) \rceil + 1 \leq 8 \log N$ and $\log N' \leq 2 \log N$.

Assume for contradiction that there are more than $\frac{k}{64\lambda}$ long paths in $\mathcal{R}'_j$. Then:

$$\sum_{Q \in \mathcal{R}'_j} |E(Q) \cap E'| \geq \frac{k}{64\lambda} \cdot d' \cdot \log^{10} N \geq \frac{k}{2^9} \cdot d' \cdot \log^9 N > 512 k\eta' \log^2 N$$

since $\lambda = \lceil \log(64N) \rceil + 1 \leq 8 \log N$, $\eta' = 2^{20} \cdot d' \cdot \log^6 N$, and N is sufficiently large. This is a contradiction since we have shown that $\sum_{Q \in \mathcal{R}'_j} |E(Q) \cap E'| \leq 512 k\eta' \log^2 N$. □

We let $\mathcal{R}_j \subseteq \mathcal{R}'_j$ be the set of all short paths, so $|\mathcal{R}_j| \geq \left(1 - \frac{1}{32\lambda}\right) k$. Since the paths in $\mathcal{R}_j$ are short, and since regular and special edges alterante on each path, the length of each such path is at most $4d' \log^{10} N$. The paths in $\mathcal{R}_j$ cause congestion at most $4\eta' \cdot \log N' \leq d' \log^8 N$, since $\log N' \leq 2 \log N$ from Inequality 10, $\eta' = 2^{20} \cdot d' \cdot \log^6 N$, and we have assumed that N is large enough. We add the paths of $\mathcal{R}_j$ to set $\mathcal{R}$, and continue to the next iteration.

Case 2: $|\mathcal{R}'_j| < \left(1 - \frac{1}{64\lambda}\right) k$

We now consider the second case, where $|\mathcal{R}'_j| < \left(1 - \frac{1}{64\lambda}\right) k$. Let $A'_j \subseteq A_j$ and $B'_j \subseteq B_j$ be the sets of vertices that do not serve as endpoints of the paths in $\mathcal{R}'_j$, so $|A'_j|, |B'_j| \geq \frac{k}{64\lambda}$. Consider the cut $(\tilde{Y}, \tilde{Y}')$ in graph H' that the algorithm for the RouteAndCut problem returns, and recall that $A'_j \subseteq \tilde{Y}$, $B'_j \subseteq \tilde{Y}'$, and additionally:

$$|E_{H'}(\tilde{Y},\tilde{Y}')| \le \frac{64\Delta'}{\eta' \log^4(|V(H')|)} + \frac{256|\mathcal{R}'_j|}{\eta'} \le \frac{512k}{\eta'} \le \frac{k}{1024d' \log^6 N},$$

since $\eta' = 2^{20} \cdot d' \cdot \log^6 N$.

Since $A'_j \subseteq \tilde{Y}$, we get that $|\tilde{Y} \cap U_\lambda| \ge \frac{k}{64\lambda}$. Next, we show that $|\tilde{Y}| \le O(k \log N)$, $|A'_j| \ge |\tilde{Y} \cap \beta| \cdot \log N$, and that $\tilde{Y}'$ must contain at least $|\beta| \cdot \left(1 - \frac{2}{\log^3 N}\right)$ vertices of β, in the following simple observation, whose proof is deferred to Section C.2 of Appendix.

Observation 6.4 *$|\tilde{Y}| \le O(k \log N)$, $|\tilde{Y}' \cap \beta| \ge \max\left\{|\beta| \cdot \left(1 - \frac{2}{\log^3 N}\right), |\beta| - \frac{2k}{\log^3 N}\right\}$, and $|A'_j| \ge |\tilde{Y} \cap \beta| \cdot \log N$.*

If Case 2 happens, then we transform the cut $(\tilde{Y},\tilde{Y}')$ in H' into either a weakly well-structured cut (A',B') in H with $|A'| \ge \frac{511|V(H)|}{512}$, $|A' \cap \beta| < \frac{2|\beta|}{\log^3 N}$, and $|E_H(A',B')| \le \frac{W}{1024d' \cdot \log^6 N}$; or into a strongly well-structured $\frac{1}{4d'}$-sparse cut (A,B) in G with $w(A), w(B) \ge \frac{k}{2^{14} \log N}$, and $w(A) \ge 1.8|\beta \cap A|$. We do so in three steps. In the first step, we obtain a weakly well-structured cut (X_1, X'_1) in H. If this cut has the required properties, namely that $|X_1| \ge \frac{511|V(H)|}{512}$, $|X_1 \cap \beta| < \frac{2|\beta|}{\log^3 N}$, and $|E_H(X_1, X'_1)| \le \frac{W}{1024d' \cdot \log^6 N}$, then we return this cut and terminate the algorithm. Otherwise, in the second step, we compute a weakly well-structured cut (X_2, X'_2) in G. Finally, in the third step, we compute the required strongly well-structured cut (A,B) in G.

Step 1: a Weakly Well-Structured Cut in H. We start by defining a cut (X_1, X'_1) in H with $X_1 = \tilde{Y} \cap V(H)$ and $X'_1 = \tilde{Y}' \cap V(H)$. Clearly, $|E_H(X_1, X'_1)| \le |E_{H'}(\tilde{Y},\tilde{Y}')| \le \frac{k}{1024d' \log^6 N}$. Moreover, $|X_1| \ge |A'_j| \ge \frac{k}{64\lambda}$. Furthermore, from Observation 6.4, we get that $|X_1| \le |\tilde{Y}| \le O(k \log N)$; $|X_1| \ge |A'_j| \ge |\tilde{Y} \cap \beta| \cdot \log N = |X_1 \cap \beta| \cdot \log N$; and $|X'_1 \cap \beta| \ge |\tilde{Y}' \cap \beta| \ge \max\left\{|\beta| \cdot \left(1 - \frac{2}{\log^3 N}\right), |\beta| - \frac{2k}{\log^3 N}\right\}$.

Next, we turn the cut (X_1, X'_1) into a weakly well-structured one. In order to do so, we consider every vertex $v \in X_1$ one by one. When vertex $v \in X_1$ is considered, we check whether there is a regular edge (v,u) that is incident to v and belongs to $E_H(X_1, X'_1)$. If so, then $v \in L'$ must hold. We then we move v from X_1 to X'_1. Notice that this move may introduce at most one new edge to the cut $E_H(X_1, X'_1)$ – the special edge that is incident to v, while at least one edge that is incident to v is removed from $E_H(X_1, X'_1)$. Therefore, throughout this transformation, $|E_H(X_1, X'_1)|$ cannot grow, and $|X_1|$ decreases by at most $|E_H(X_1, X'_1)| \le \frac{k}{d' \cdot \log^6 N}$. Since, at the beginning of Step 1, $|X_1| \ge \frac{k}{64\lambda} \ge \frac{k}{256 \log N}$ held, at the end of this step, $|X_1| \ge \frac{k}{512 \log N}$ holds, and moreover $|X_1| \ge \frac{|X_1 \cap \beta| \cdot \log N}{2}$. Since we did not remove any vertices from X'_1, it is still the case that $|X'_1 \cap \beta| \ge \max\left\{|\beta| \cdot \left(1 - \frac{2}{\log^3 N}\right), |\beta| - \frac{2k}{\log^3 N}\right\}$. It is easy to verify that every edge in $E_H(X_1, X'_1)$ is a special edge, and so (X_1, X'_1) is a weakly well-structured in H. Notice that the time required for this step is bounded by $O(\mathrm{Vol}_H(X_1)) \le O(|X_1| \cdot W) \le O(k \cdot W \cdot \log N)$.

We now consider two cases. The first case happens if $|X'_1| < \frac{k}{512}$. Then we are guaranteed that:

$$|E_H(X_1, X'_1)| \le \frac{k}{1024d' \cdot \log^6 N} \le \frac{|V(H)|}{1024d' \cdot \log^6 N}.$$

Additionally, we get that $|X'_1| \le \frac{k}{512} \le \frac{|V(H)|}{512}$, and $|X'_1 \cap \beta| \ge |\tilde{Y}' \cap \beta| \ge |\beta| \cdot \left(1 - \frac{2}{\log^3 N}\right)$. Therefore, $|X_1| \ge \frac{511|V(H)|}{512}$, and $|X_1 \cap \beta| < \frac{2|\beta|}{\log^3 N}$. We return cut (X_1, X'_1) in H and terminate the algorithm.

From now on, we assume that $|X_1'| \geq \frac{k}{512}$. We denote $\varphi = \frac{1}{d'}$. Since $|E_H(X_1, X_1')| \leq \frac{k}{d' \log^6 N}$, while $|X_1| \geq \frac{k}{512 \log N}$, the sparsity of the cut (X_1, X_1') in H is at most $\frac{1}{d' \log^4 N} = \frac{\varphi}{\log^4 N}$. Moreover, (X_1, X_1') is a weakly well-structured cut in H and $|X_1 \cap \beta| < \frac{2k}{\log^3 N}$.

Step 2: a Weakly Well-Structured Cut in G. Next, we turn the cut (X_1, X_1') in H into a cut (X_2, X_2') in G, in a natural way: we let $X_2 = X_1 \cap V(G)$, and $X_2' = X_2 \cap V(G)$. Recall that the length of every edge in G is at most $d' \leq \frac{1}{\varphi}$. From Observation 4.4, (X_2, X_2') is a weakly well-structured cut in G of sparsity at most $\frac{8\varphi}{\log^4 N}$, with $w(X_2) \geq \frac{|X_1|}{8} \geq \frac{k}{2^{12} \log N}$ and $w(X_2') \geq \frac{|X_1'|}{8} \geq \frac{k}{2^{12} \log N}$. The time required to compute the cut (X_2, X_2') in G given the cut (X_1, X_1') in H is bounded by $O(|V(H)|) \leq O(W)$. Clearly, $|X_2 \cap \beta| \leq |X_1 \cap \beta| < \frac{2k}{\log^3 N}$.

Step 3: a Strongly Well-Structured Cut in G. Recall that for every special edge $e \in E(G)$, $\ell(e) \leq d' \leq \frac{n}{8}$ (from the definition of the input to Procedure ReconnectLayer), while $\sum_{e \in E(G)} \ell(e) \geq \frac{n}{2}$. We can now apply the algorithm from Observation 4.5 to cut (X_2, X_2') compute a strongly well-structured cut (A, B) in G with $w(A) \geq \frac{w(X_2)}{4} \geq \frac{k}{2^{14} \log N}$ and $w(B) \geq \frac{w(X_2')}{4} \geq \frac{k}{2^{14} \log N}$, whose sparsity is at most $\frac{16\varphi}{\log^4 N} \leq \frac{1}{4d'}$.

We now show that $w(A) \geq 1.8|\beta \cap A|$. Recall that the algorithm from Observation 4.5 starts with the cut $(A, B) = (X_2, X_2')$, and considers every special edge $e = (u, v) \in E_G(B, A)$ in turn. For each such edge, it either moves u from B to A, or it moves v from A to B. We partition the set $A \cap \beta$ of vertices into three subsets: set β_1 contains all vertices $v \in A \cap \beta$ that lied in X_2; recall that $|\beta_1| \leq |X_2 \cap \beta| \leq \frac{2k}{\log^3 N}$. We now define the remaining two sets β_2 and β_3. Each vertex $v \in (\beta \cap A) \setminus \beta_1$ was added to set A by the algorithm from Observation 4.5. Therefore, for each such vertex v, there is a special edge $(v, u) \in E_G(X_2', X_2)$. If $u \in \beta$ holds (in which case $u \in \beta_1$ holds), we add v to β_2, and otherwise we add it to β_3. It is easy to verify that $|\beta_2| \leq |\beta_1| \leq \frac{2k}{\log^3 N}$. Consider now a vertex $v \in \beta_3$, and the corresponding special edge $e = (v, u)$. Since $w(v), w(u) \geq 1$, it is easy to see that $|\beta_3| \leq w(A)/2$. Overall, we get that:

$$|\beta \cap A| = |\beta_1| + |\beta_2| + |\beta_3| \leq \frac{4k}{\log^3 N} + \frac{w(A)}{2} \leq 0.51 w(A).$$

Therefore, $w(A) \geq 1.8|\beta \cap A|$, as required.

The running time of the algorithm from Observation 4.5 is bounded by $O(\text{Vol}_G(X_2)) \leq O(\text{Vol}_H(X_1)) \leq O(|X_1| \cdot |V(H)|) \leq O(k \cdot W \cdot \log N)$. We return the cut (A, B) and terminate the algorithm.

Overall, the total time we have spent on iteration j when Case 2 happens, excluding the time required to execute Algorithm $\mathcal{A}$, is bounded by $O(k \cdot W \cdot \log N)$. This concludes the description and the analysis of a single iteration.

6.2.4 Constructing a New Layered Connectivity Data Structure.

If any iteration terminated with a "FAIL", then the algorithm terminates with a "FAIL". Since the number of iterations is bounded by $4\lambda \leq 32 \log N$, and the probability that a single iteration terminates with a "FAIL" is bounded by $2^{-\log^3 N}$, we get that the overall probability that our algorithm implementing procedure ReconnectLayer terminates with a "FAIL" is bounded by $\frac{1}{N^{10}}$. We assume from now on that no iteration terminated with a " FAIL". If, in any iteration, Case 2 occurred, then the algorithm terminated with a cut.

We assume from now on that neither of the 4λ iterations terminated with a "FAIL", and that, in each iteration, Case 1 occurred. Therefore, for all $1 \le j \le 4\lambda$, we have computed a collection $\mathcal{R}_j$ of paths, with $|\mathcal{R}_j| \ge \left(1 - \frac{1}{32\lambda}\right) k$, such that the length of every path in $\mathcal{R}_j$ is at most $4d' \log^{10} N$. Let $\mathcal{R} = \bigcup_{j=1}^{4\lambda} \mathcal{R}_j$. Since, for all $1 \le j \le 4\lambda$, the paths in $\mathcal{R}_j$ cause congestion at most $d' \log^8 N$, and $\lambda = \lceil \log(64N) \rceil + 1 \le 4 \log N$, we get that the paths in $\mathcal{R}$ cause congestion at most $4d' \log^9 N$. For every vertex $v \in U_\lambda$, let $\mathcal{R}(v) \subseteq \mathcal{R}$ be the set of paths that originate at v. Clearly, $|\mathcal{R}(v)| \le 4\lambda$ holds.

We say that a vertex $v \in U_\lambda$ is *good* if $|\mathcal{R}(v)| \ge 2\lambda$, and we say that it is bad otherwise. We claim that at most $\frac{k}{4\lambda}$ vertices of U_λ may be bad. Indeed, let Π be the collection of all pairs (v, j), where $v \in U_\lambda$ and $1 \le j \le 4\lambda$, such that no path of $\mathcal{R}_j$ originates at v. Since, for all $1 \le j \le 4\lambda$, $|\mathcal{R}_j| \ge \left(1 - \frac{1}{32\lambda}\right) k$, we get that $|\Pi| \le \frac{k}{32\lambda} \cdot 4\lambda = \frac{k}{8}$. However, if more than $\frac{k}{4\lambda}$ vertices of U_λ are bad, then $|\Pi| > \frac{k}{4\lambda} \cdot 2\lambda \ge \frac{k}{2}$, a contradiction. We let $U' \subseteq U_\lambda$ be the set of all good vertices, so $|U'| \ge \left(1 - \frac{1}{4\lambda}\right) k$.

Let $\tilde{\mathcal{R}} \subseteq \mathcal{R}$ be a set of path that contains, for every vertex $v \in U'$, exactly 2λ paths that originate at v. From our construction, the paths in $\tilde{\mathcal{R}}$ have length at most $4d' \log^{10} N$ each, and they cause congestion at most $4d' \log^9 N$. Additionally, every vertex $x \in \beta$, the number of paths in $\mathcal{R}$ that terminate at x is bounded by:

$$(4\lambda) \cdot M \le \frac{4\lambda \cdot W \log^3 N}{|\beta|} \le \frac{\lambda \rho}{4},$$

since $M = \frac{W \cdot \log^3 N}{|\beta|}$ and $\frac{W \cdot \log^3 N}{|\beta|} \le \frac{\rho}{16}$ from the problem definition. Our algorithm also ensures that vertex $y \in U^{<i} \setminus \beta$ may serve as an endpoint of at most one such path. Consider now some vertex $x \in \beta$, and let $\mathcal{D}^{<i}(x)$ be the set of all descendant vertices of x that belong to layers $U_1, \ldots, U_{i-1}$, including vertex x itself. From Property L2 of Layered Connectivity Data Structure, $|\mathcal{D}^{<i}(x)| \le i \cdot \rho$. Let $\tilde{\mathcal{R}}^x \subseteq \tilde{\mathcal{R}}$ denote the collection of all paths $Q \in \tilde{\mathcal{R}}$, such that Q terminates either at one of the vertices in $\mathcal{D}^{<i}(x)$. Then $|\tilde{\mathcal{R}}^x| \le \frac{\lambda\rho}{4} + \lambda \cdot \rho \le 1.5\lambda \cdot \rho$.

Notice that for every path $Q \in \tilde{\mathcal{R}}$, if $v \in U^{<i}$ is the last endpoint of Q, then we can compute a vertex $x \in \beta$ with $v \in \mathcal{D}^{<i}(x)$ in time $O(1)$, using the current Layered Connectivity Data Structure. We compute the collection $\beta' \subseteq \beta$ of vertices, where a vertex $x \in \beta'$ if and only if some path in $\tilde{R}$ terminates at a vertex of $\mathcal{D}^{<i}(x)$. This set of vertices can be computed in time $O(|\tilde{R}|) \le O(\lambda k) \le O(k \log N)$.

Let $V' \subseteq U^{<i}$ be the set of vertices that serve as endpoints of the paths in $\tilde{\mathcal{R}}$. From our construction, every vertex $v \in V' \setminus \beta$ serves as an endpoint of exactly one such path. Next, we construct an auxiliary directed flow network $\tilde{H}$. We start by letting $V(\tilde{H}) = U' \cup V' \cup \beta'$. For every path $Q \in \tilde{R}$, if v is the first endpoint of Q, and v' is the last endpoint of Q then we add a directed edge $e = (v, x)$ to the flow network, whose capacity is 1. We say that edge e *corresponds to the path* Q. For every vertex $u \in V'$, if $x \in \beta'$ is the ancestor of v, then we add an edge (u, x) of capacity 1 to the graph. Additionally, we add a source vertex s, and connect it to every vertex in U' with an edge of capacity 1, and a destination vertex t. Every vertex $x \in \beta'$ connects to t with an edge of capacity ρ. This concludes the description of the directed flow network $\tilde{H}$. We claim that the value of the maximum s-t flow in this network is $|U'|$. Indeed, since the total capacity of all edges incident to s is $|U'|$, the value of the maximum s-t flow cannot be greater than $|U'|$. Consider now the following fractional flow in $\tilde{H}$: we send one flow unit on every edge incident to s, and, for every path $Q \in \tilde{\mathcal{R}}$, we send $\frac{1}{2\lambda}$ flow units on its corresponding edge $(v, u) \in E(\tilde{H})$, and the same amount of flow on edge (u, x), where $x \in \beta'$ is the ancestor of u. Since, for every vertex $v \in U'$, set $\tilde{\mathcal{R}}$ contains exactly 2λ paths originating at v, the total amount of flow leaving v is exactly 1. Since every vertex $u \in V'$ is an endpoint of at most one

path in $\tilde{\mathcal{R}}$, if $x \in \beta'$ is the ancestor of u, then edge (u,x) carries at most 1 flow unit.

Consider now some vertex $x \in \beta'$, and recall that $|\tilde{\mathcal{R}}^x| \leq 1.5\lambda \cdot \rho$. Since, for every path $Q \in \tilde{\mathcal{R}}^x$, the corresponding edge carries $\frac{1}{2\lambda}$ flow units, the total amount of flow entering x is at most $\frac{1.5\lambda\cdot\rho}{2\lambda} \leq \rho$. We set the flow on the edge (x,t) to be equal to the total amount of flow entering x. Overall, we obtain a valid s-t flow in $\tilde{H}$, whose value is $|U'|$.

Next, we employ the standard Ford-Fulkerson algorithm in order to compute an integral flow of value $|U'|$ in $\tilde{H}$. We start by setting the flow on every edge of $\tilde{H}$ to 0, and then perform at most $|U'| \leq k$ iterations. In every iteration, we compute a residual flow network, along with an augmenting path, and then modify the current flow accordingly. Clearly, every iteration can be executed in time $O(|E(\tilde{H})|) \leq O(k \log N)$, while the number of iterations is bounded by k. Altogether, the running time of the algorithm that computes the flow network $\tilde{H}$, and computes a maximum integral s-t flow in it is bounded by $O(k^2 \log N) \leq O(k \cdot W \log N)$.

We are now ready to construct a new Layered Connectivity Data Structure. The sets $U_0, \ldots, U_{i-1}$ of vertices, and the sets $\mathcal{Q}_1, \ldots, \mathcal{Q}_{i-1}$ of paths remain unchanged. For all $i < i' < \lambda$, we set $U_{i'} = \emptyset$. We also set $U_i = U'$ and $U_\lambda = V(H) \setminus \left(\bigcup_{i'=0}^{i} U_{i'}\right)$. Since $|U'| \geq \left(1 - \frac{1}{4\lambda}\right) k$, we get that $|U_\lambda| \leq \frac{k}{4\lambda}$. For every vertex $v \in U'$, we consider the unique edge e of $\tilde{H}$ that leaves the vertex v and carries one flow unit, and we let $Q \in \tilde{\mathcal{R}}$ be the path corresponding to e. We then set $K(v) = Q$, and we let $\mathcal{Q}_i = \{K(v) \mid v \in U_i\}$. Since, for every vertex $x \in \beta'$, the capacity of the edge (x,t) in $\tilde{H}$ is ρ, this guarantees that the number of descendant-vertices of x in U_i is bounded by ρ, as required. For every vertex $u \in V'$, if $x \in \beta'$ is its ancestor, then the capacity of the edge (u,x) is 1, so u may serve as an endpoint in at most 1 path in $\mathcal{Q}_i$. Additionally, our construction guarantees that the paths in $\mathcal{Q}_i$ have length at most $4d' \log^9 N$ each, and they cause edge-congestion at most $4d' \log^9 N$ in H, as required. Therefore, we obtain a valid Layered Connectivity Data Structure.

6.2.5 Analysis of the Running Time

Recall that the time required to construct the adjacency-list representation of graph H' from the adjacency-list of H is bounded by $O(W \log^3 N)$. The algorithm consists of $4\lambda = O(\log N)$ iterations. In every iteration, we apply Algorithm $\mathcal{A}$ for the r-restricted RouteAndCut problem to instance $(H', A_j, B_j, \Delta', \eta', N')$ at most $O(\log^3 N)$ times. Recall that $|V(H')| \leq O(W \cdot \log^3 N)$, and $|V(H') \setminus B_j| \leq 4\lambda k \leq O(k \log N)$. Therefore, the running time of each execution of Algorithm $\mathcal{A}$ is bounded by:

$$
\begin{aligned}
c_1 \cdot |V(H')| \cdot (|V(H')| - |B_j|) \cdot 2^{c_2\sqrt{\log N'}} \cdot (\log N')^{16c_2(r-1)+8c_2} \\
&\leq O\left(c_1 \cdot W \cdot k \cdot 2^{c_2\sqrt{\log N'}} \cdot (\log N')^{16c_2(r-1)+8c_2+4}\right) \\
&\leq O\left(c_1 \cdot W \cdot k \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+10} \cdot \left(1 + \frac{1}{64c_2 r}\right)^{16c_2(r-1)+8c_2+4}\right) \\
&\leq O\left(c_1 \cdot W \cdot k \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+10}\right)
\end{aligned}
$$

(We have used the fact that, from Inequality 11, $2^{\sqrt{\log N'}} \leq 2^{\sqrt{\log N}} \cdot \log^6 N$, and from Inequality 10, $\log N' \leq \left(1 + \frac{1}{64c_2 r}\right) \cdot \log N$.

The time spent on executing Algorithm $\mathcal{A}$ in all $4\lambda = O(\log N)$ iterations is then bounded by:

$$O\left(c_1 \cdot W \cdot k \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+14}\right).$$

If the algorithm terminates with a cut, then we may need to invest additional $O(k \cdot W \cdot \log N)$ time in order to convert the cut returned by Algorithm $\mathcal{A}$ into a desired well structured cut in H or in G. Lastly, if the algorithm terminates with a new Layered Connectivity Data Structure, then the additional time that is required in order to compute the flow network $\tilde{H}$, the maximum integral s-t flow in it, and then compute the corresponding set $\mathcal{Q}_i$ of paths, is bounded by $O(k \cdot W \cdot \log N)$. Overall, we get that the running time of the algorithm is bounded by:

$$O\left(c_1 \cdot W \cdot k \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+14}\right).$$

6.2.6 Special Case: $k < 2^{20} \cdot d' \cdot \log^6 N$.

It now remains to consider a special case where $k < 2^{20} \cdot d' \cdot \log^6 N$. In this case, we employ the following simple algorithm. We start with $U' = \emptyset$, $\mathcal{R} = \emptyset$, and $T = V(H) \setminus U_\lambda$. In every iteration, we will add a single path to $\mathcal{R}$, connecting some vertex in U_λ to a vertex of T. For every vertex $x \in \beta$, we maintain a counter n_x, that counts the number of paths in $\mathcal{R}$ that terminate at the vertices of $\bigcup_{i'<i} \mathcal{D}_{i'}(x)$. Once this number reaches ρ, we remove vertex x, and all vertices in $\bigcup_{i'<i} \mathcal{D}_{i'}(x)$ from T. We also delete from T any vertex that serves as an endpoint of a path in $\mathcal{R}$, except for the vertices that lie in β. We initialize $n_x = 0$ for all $x \in \beta$. We will ensure that $|\mathcal{R}| \leq k < 2^{20} \cdot d' \cdot \log^6 N$. This ensures that the total number of vertices $x \in \beta$ for which $n_x = \rho$ holds at the end of the algorithm is bounded by $\frac{k}{\rho}$. From Requirement L2 of the Layered Connectivity Data Structure, for each such vertex $x \in \beta$, $|\mathcal{D}(x)| \leq \lambda \cdot \rho$, and so the total number of vertices that are ever deleted from T over the course of the algorithm is bounded by $\frac{k}{\rho} \cdot \lambda \cdot \rho + k \leq 2\lambda \cdot k \leq 8k \log N$. Therefore, throughout the algorithm, $|V(H) \setminus T| \leq k + 8k \log N \leq 9k \log N \leq 2^{24} \cdot d' \cdot \log^7 N$ holds.

We consider every vertex $v \in U_\lambda$ one by one. When vertex v is considered, we either compute a path $K(v)$ connecting v to a vertex of T, such that no inner vertices on $K(v)$ lie in T; or we compute a set $S_v \subseteq V(H) \setminus T$ of vertices with no outgoing edges, that contains v. In the former case, the length of $K(v)$ is at guaranteed to be at most $|V(H) \setminus T| \leq 2^{14} \cdot d' \cdot \log^7 N \leq 4d' \log^{10} N$. We add vertex v to set U', add path $K(v)$ to set $\mathcal{R}$, and, if $u \in T$ is the other endpoint of path $K(v)$, and $x \in \beta$ is the vertex for which $u \in \mathcal{D}(x)$, we increase the counter n_x, and, if $u \notin \beta$, delete u from T. If n_x reaches ρ, then we delete from T all vertices of $\bigcup_{i'<i} \mathcal{D}_{i'}(x)$. We then continue to the next iteration. In the latter case, if we obtain a set $S_v \subseteq V(H) \setminus T$ of vertices with no outgoing edges with $v \in S_v$, we just continue to the next iteration.

If, at the end of this procedure, $|U'| \geq \left(1 - \frac{1}{4\lambda}\right) k$, then we obtain a new Layered Connectivity Data Structure by keeping the layers $U_0, \ldots, U_{i-1}$ unchanged, together with the corresponding collections $\mathcal{Q}_1, \ldots, \mathcal{Q}_{i-1}$ of paths, setting $U_i = U'$ and $\mathcal{Q}_i = \mathcal{R} = \{K(v) \mid v \in U'\}$. Note that the paths in $\mathcal{Q}_i$ cause congestion at most $k \leq 2^{20} \cdot d' \cdot \log^6 N \leq 4d' \log^9 N$. We also let U_λ contain all vertices that originally lied in U_λ and were not added to U', and, for all $i < i' < \lambda$, we set $U_{i'} = \emptyset$. From our construction, for every vertex $x \in \beta$, $|\mathcal{D}_i(x)| \leq \rho$ holds, as required, and every vertex $v \in U^{<i}$ serves as endpoint of at most one path in $\mathcal{Q}_i$. It is immediate to verify that we obtain a valid Layered Connectivity Data Structure, with $|U_\lambda| \leq \frac{k}{4\lambda}$.

Assume now that, at the end of the algorithm, $|U'| < \left(1 - \frac{1}{4\lambda}\right) k$ holds. Consider the cut $(\tilde{Y}, \tilde{Y}')$ in H, where $\tilde{Y} = \bigcup_{v \in U_\lambda \setminus U'} S_v$ and $\tilde{Y}' = V(H) \setminus \tilde{Y}$. Clearly, $E_H(\tilde{Y}, \tilde{Y}') = \emptyset$. Moreover, from our construction, $|\tilde{Y}| \geq |U_\lambda \setminus U'| \geq \frac{k}{4\lambda} \geq \frac{k}{512 \log N}$. At the same time, $\tilde{Y} \subseteq V(H) \setminus T$, so $|\tilde{Y}| \leq |V(H) \setminus T| \leq$

$2^{24} \cdot d' \cdot \log^7 N$. Since the total number of vertices $x \in \beta$ with $n_x = \rho$ at the end of the algorithm is bounded by $\frac{k}{\rho} \le \frac{W}{\rho}$, while, from the problem definition, $\frac{W \cdot \log^3 N}{|\beta|} \le \frac{\rho}{16}$, we get that $W \le \frac{|\beta| \cdot \rho}{16 \log^3 N}$, and $|\tilde{Y}' \cap \beta| \ge |\beta| - \frac{W}{\rho} \ge |\beta| \left(1 - \frac{2}{\log^3 N}\right)$.

Since $E_H(\tilde{Y}, \tilde{Y}') = \emptyset$, cut $(\tilde{Y}, \tilde{Y}')$ is weakly well-structured in H, and it is $\frac{1}{d' \log^4 N}$-sparse. If $|\tilde{Y}'| < \frac{k}{512}$, then we return cut $(\tilde{Y}, \tilde{Y}')$ in H, that is guaranteed to have all required properties. Otherwise, we transform cut $(\tilde{Y}, \tilde{Y}')$ into a a strongly well-structured $\frac{1}{4d'}$-sparse cut (A, B) in G with $w(A), w(B) \ge \frac{k}{2^{14} \log N}$ and $w(A) \ge 1.8|\beta \cap A|$, exactly as before, and output this cut as the outcome of the algorithm.

We now provide an algorithm to efficiently implement the above procedure. Throughout the algorithm, we denote $Z = V(H) \setminus T$. Each vertex $v \in Z$ is classified as either *marked* or *unmarked*; a vertex $v \in Z$ is marked only if we have correctly established that no path connects v to a vertex of T. Notice that vertices may leave T, but they cannot join T. Therefore, if, at some time τ during the algorithm, no path conencts v to vertices of T, then this remains so for the remainder of the algortihm. For every vertex $v \in Z$ that is unmarked, we maintain a heap $h(v)$ that contains all unmarked vertices $u \in Z$ with edge $(v, u) \in E(H)$, and also all vertices $u' \in T$ with edge $(v, u') \in E(H)$.

At the beginning of the algorithm, $Z = U_\lambda$, and all vertices in Z are unmarked. Notice that the time that is required in order to construct the initial heaps $h(v)$ for vertices $v \in Z$ is bounded by $O(|U_\lambda| \cdot |V(H)|) \le O(k \cdot W)$. Next, we perform iterations. In every iteration, we select an arbitrary vertex $v \in Z$, and perform a DFS, starting from v, in the graph that is obtained from H by deleting the marked vertices, until a vertex of T is reached. In other words, when the DFS search arrives at some vertex z, it only explores edges (z, u) with $u \in h(z)$. Whenever the DFS search backtracks from a vertex a, we know that no path connects a to a vertex of T, so we mark a. We then examine all edges $(b, a) \in E(H)$ with $b \in Z$, and delete a from the corresponding heap $h(b)$. The time that the DFS spent on exploring vertex a, on marking it, and on deleting it from the heaps of vertices $b \in Z$ with $(b, a) \in E(H)$ is *charged* to vertex a itself. Excluding the time charged to the newly marked vertices, the DFS search from a single vertex v takes time $O(|Z|) = O(k \log N)$. If, as the result of the search, we arrive at a vertex of T, then we obtain the desired path $K(v)$. Otherwise, vertex v becomes marked as well, and we process it like all other marked vertices. Notice that the number of such DFS searches that our algorithm performs is bounded by $|U_\lambda| \le k$, and so the total time spent on the DFS searches, excluding the time charged to the marked vertices, is bounded by $O(k^2 \log N) \le O(k \cdot W \cdot \log N)$.

Recall that, throughout the algorithm, $|Z| = |V(H) \setminus T| \le O(k \log N)$ holds. Every vertex in Z may be marked at most once. When a vertex $a \in Z$ is marked, we need to consider all vertices $b \in Z$ with $(b, a) \in E(H)$, and delete a from the heap $h(b)$. Since $|Z| \le O(k \log N)$, we spend $O(k \log N)$ to process each newly marked vertex, and $O(k^2 \log^2 N) \le O(k \cdot W \log^2 N)$ time on all marked vertices throughout the algorithm.

Once every vertex of U_λ is either added to U' or is marked, the algorithm terminates. The set $\tilde{Y}$ of vertices can then be constructed by simply including all marked vertices of Z in it. From the above discussion, the total running time of the algorithm for this special case is $O(k \cdot W \cdot \log^2 N)$. This completes the proof of Claim 6.1.

6.3 ConnectToCenters Problem

Suppose we are given a perfect well-structured graph G with a proper assignment $\ell(e)$ of lengths to its edges, together with a parameter $d' \ge 1$, and a subset $\beta \subseteq V(G)$ of vertices of G. Assume further that G undergoes an online sequence of updates of the following two kinds: (i) double the length of a special edge $e \in E(G)$; and (ii) delete a vertex $v \in \beta$ from the set β. Intuitively, the purpose of the

ConnectToCenters problem is to maintain the Layered Connectivity Data Structure in G with $U_\lambda = \emptyset$, except that we slightly relax Requirement L1, and allow the lengths of the paths in $\bigcup_{i=1}^{\lambda-1} \mathcal{Q}_i$ to be bounded by $16d' \log^{10} N$, instead of $4d' \log^{10} N$. The parameter ρ corresponding to the data structure is $\rho = \frac{2^{12} \cdot W^0 \cdot \log^3 N}{|\beta|}$, where W^0 is the initial total weight of all vertices of G. The algorithm is allowed, at any time, to return a strongly well-structured $\frac{1}{4d'}$-sparse cut (A, B) in G with $w(A) \geq 1.8|\beta \cap A|$. Following that, if $w(A) \leq w(B)$, then the vertices of A, and the endpoints of the edges in $E_G(A, B)$ are deleted from G, while otherwise the vertices of B, and the endpoints of the edges in $E_G(A, B)$ are deleted from G. In either case, the algorithm must continue with the resulting graph. The algorithm is also allowed, at any time, to return a weakly well-structured cut (A', B') in graph H – the subdivided graph corresponding to G – with $|A'| \geq \frac{511|V(H)|}{512}$, $|A' \cap \beta| < \frac{2|\beta|}{\log^3 N}$, and $|E_H(A', B')| \leq \frac{|V(H)|}{1024d' \cdot \log^6 N}$, after which the algorithm terminates. It is also allowed, at any time, to terminate with a "FAIL", but we require that the probability that this ever happens is bounded by $1/N^7$. We model this algorithm as consisting of a number of phases, where every phase consists of a number of iterations. In every iteration, the algorithm must either (i) produce a valid Layered Connectivity Data Structure for the current graph G with $U_\lambda = \emptyset$ (with the relaxed Requirement L1), after which the phase terminates; or (ii) produce a strongly well-structured $\frac{1}{4d'}$-sparse cut (A, B) in G, after which G is updated as described above, and some vertices may be deleted from β; or (iii) produce a weakly well-structured cut (A', B') in graph H, with $|A'| \geq \frac{511|V(H)|}{512}$, $|A' \cap \beta| < \frac{2|\beta|}{\log^3 N}$, and $|E_H(A', B')| \leq \frac{|V(H)|}{1024d' \cdot \log^6 N}$, after which the algorithm terminates. At the end of each phase, the algorithm is given a collection of special edges of G whose lengths are doubled, and a collection of vertices that are deleted from β, and then a new phase begins. The algorithm only needs to continue as long as $|\beta|$ remains sufficiently large (it most contain at least 1/4 of the vertices from the original set β), and as long as the total increase in the lengths of the edges is not too large. We now define the total increase of edge lengths formally, followed by a formal defintion of the ConnectToCenters problem.

Definition 6.5 (Total increase in edge lengths) *Let G be a perfect well-structured graph with a proper assignment $\ell(e)$ of lengths to its edges, that undergoes an online sequence of updates, consisting of the following operations: (i) double the length of a special edge $e \in E(G)$; and (ii) delete a vertex from $V(G)$, together with its incident edges. Consider any time τ during the update sequence, and denote by E^0 the set of special edges in the initial graph G. For each edge $e \in E^0$, let $\ell^0(e)$ be the length of e at the beginning of the algorithm, and let $\ell^\tau(e)$ be the length of e at time τ if $e \in E(G)$ at time τ; or the length of e just before it was deleted from τ otherwise. Then the total increase in edge lengths at time τ is: $\sum_{e \in E^0} \left(\ell^\tau(e) - \ell^0(e)\right)$.*

We are now ready to define the ConnectToCenters problem.

Definition 6.6 (ConnectToCenters problem) *The input to the* ConnectToCenters *problem consists of a perfect well-structured graph $G = (L, R, E)$ with $|V(G)| = n$, and a proper assignment $\ell(e)$ of lengths to the edges $e \in E$, that are given in the adjacency list representation, together with a subdivided graph H corresponding to G in the adjacency-list representation. Let W^0 denote the weight of all vertices in this initial graph G. Additionally, we are given a parameter $N \geq \max\left\{n, W^0/8\right\}$ that is greater than a large enough constant, and parameters $d' \geq 1$, and $1 \leq r \leq \left\lceil \sqrt{\log N} \right\rceil$, such that $d' \leq \min\left\{\frac{2^{r\sqrt{\log N}}}{\log^8 N}, \frac{n}{8}\right\}$. Lastly, we are given a set $\beta \subseteq V(H)$ of vertices, with $|\beta| \geq \frac{W^0}{256d'}$. We denote by β^0 the initial set β.*

The algorithm consists of a number of phases, and every phase is partitioned into iterations. In every iteration, the algorithm is required to return one of the following:

O1. either a valid Layered Connectivity Data Structure *in the current graph G with the parameter*

$\rho = \frac{2^{12} \cdot W^0 \cdot \log^3 N}{|\beta^0|}$, in which Requirement L1 is relaxed to allow the lengths of the paths in $\bigcup_{i=1}^{\lambda-1} \mathcal{Q}_i$ to be bounded by $16d' \log^{10} N$ and $U_\lambda = \emptyset$;

O2. or a strongly well-structured $\frac{1}{d'}$-sparse cut (A, B) in G with $w(A) \geq 1.8|\beta \cap A|$;

O3. or a weakly well-structured cut (A', B') in graph H, with $|A'| \geq \frac{511|V(H)|}{512}$, $|A' \cap \beta| < \frac{2|\beta|}{\log^3 N}$, and $|E_H(A', B')| \leq \frac{|V(H)|}{1024d' \cdot \log^6 N}$.

In the case of Outcome O3, the algorithm terminates. In the case of Outcome O2, the endpoints of the edges in $E_G(A, B)$ are deleted from G. Additionally, if $w(A) \leq w(B)$, then the vertices of A are deleted from G, and otherwise the vertices of B are deleted from G. All the vertices of β that are deleted from G are also deleted from β, along with some additional vertices that may be deleted from β (given by the adversary). In case of Outcome O1, the current phase terminates, and the algorithm is given a collection of special edges of G, whose lengths are doubled, together with a list of vertices that are deleted from β.

Once the total increase in the edge lengths becomes greater than $8n$, or at least $3/4$ of vertices in the initial set β are deleted from β, the algorithm terminates. The algorithm may also return "FAIL" and terminate at any time, but we require that the probability that this ever happens is bounded by $\frac{1}{N^7}$.

Next, we provide an algorithm for the ConnectToCenters problem, that relies on the algorithm for the r-restricted RouteAndCut problem, whose existence is assumed in Theorem 4.12. We then show that the algorithm for the r-restricted MaintainSpecialCluster problem with the properties required in Theorem 4.12 follows immediately from the algorithm for the ConnectToCenters problem. This algorithm will also be used as one of the main ingredients in our algorithm for the MaintainCluster problem, that will complete the proof of Theorem 4.12.

6.4 An Algorithm for the ConnectToCenters Problem

In this subsection, we provide an algorithm for the ConnectToCenters problem, that is summarized in the following theorem.

Theorem 6.7 *Suppose that, for some parameter $r \geq 1$, there exists a randomized algorithm for the r-restricted* RouteAndCut *problem, that, given as an input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most $c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2}$. Then there is a randomized algorithm for the* ConnectToCenters *problem, for the same value of parameter r, whose running time is at most:*

$$O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+26}\right),$$

where Λ is the total increase in the edge lengths over the course of the algorithm, β' is the set of all vertices deleted from β over the course of the algorithm (including the vertices that are deleted from H following updates to graph G), and W^0 is the total initial weight of the vertices in the input graph G.

The remainder of this subsection is dedicated to the proof of Theorem 6.7. The algorithm is quite straightforward: in every iteration we simply apply the algorithm for the ReconnectLayer problem from Claim 6.1 to the current graph G, and the current Layered Connectivity Data Structure. The only subtlety is that the algorithm requires that the length of every edge in G is at least d'. If this is not the case, and $e = (u, v)$ is a special edge of length greater than d', then we can define a strongly well-structured cut (A, B) in G with $A = \{u\}$ and $B = V(G) \setminus \{u\}$, whose sparsity is at least $1/d'$ (since $w(u), w(v) \geq d'$), and return this cut. We also need to verify that, whenever we apply the algorithm from Claim 6.1 to the current graph G, we indeed obtain a valid instance of the ReconnectLayer problem. The main difficulty is in the analysis of the running time of this algorithm, that we discuss below.

6.4.1 The Data Structures

Recall that we are given as input a perfect well-structured graph $G = (L, R, E)$ with $|V(G)| = n$, and a proper assignment $\ell(e)$ of lengths to the edges $e \in E$, that are given in the adjacency list representation, together with a subdivided graph $H = (L', R', E')$ corresponding to G in the adjacency-list representation. We denote by W^0 the weight of all vertices in this initial graph G. Recall that we are also given a parameter $N \geq \max\left\{n, W^0/8\right\}$ that is greater than a large enough constant, and parameters $d' \geq 1$, and $1 \leq r \leq \left\lceil\sqrt{\log N}\right\rceil$, such that $d' \leq \min\left\{\frac{2^{r\sqrt{\log N}}}{\log^8 N}, \frac{n}{8}\right\}$. We also denote by β^0 the initial set $\beta \subseteq V(H)$ of vertices that is given as part of the problem input; recall that $|\beta^0| \geq \frac{W^0}{256d'}$. Throughout, we use the parameter $\rho = \frac{2^{12} \cdot W^0 \cdot \log^3 N}{|\beta^0|}$.

Recall that graph H is obtained from G by replacing every special edge $e = (u, v)$ with a new path, that we denote by $P^*(e)$, and call *the path representing* e *in* H. The number of special edges on $P^*(e)$ is exactly $\ell(e)$. Therefore, $|V(H)| = \sum_{v \in V(G)} w(v)$, and initially, $|V(H)| = W^0$.

Assume that at some time during the algorithm's execution, the length $\ell(e)$ of a special edge $e \in E(G)$ is doubled. Then for every special edge $e' = (x, y)$ on path $P^*(e)$, we replace the edge by a path (x, x', y', y), where x', y' are new vertices, edges (x, x') and (y', y) are special, and edge (x, y') is regular; we call this process the *splitting* of edge e'. Notice that, since (x, y) is a special edge of H, $x \in R'$ and $y \in L'$ must hold. We add x' to L' and y' to R'. It is easy to verify that H remains the subdivided graph of G after this update. Since the total increase in the lengths of the edges of G over the course of the algorithm is bounded by Λ, the total time required to maintain graph H in the adjacency-list representation, once the representation of the initial graph H is given, is bounded by $O(\Lambda)$.

Our algorithm maintains a valid Layered Connectivity Data Structure for the current graph G, with the parameter ρ defined above, and with Requirement L1 relaxed to allow the lengths of the paths in $\bigcup_{i=1}^{\lambda-1} \mathcal{Q}_i$ to be bounded by $16d' \log^{10} N$. However, whenever a new path is added to set $\bigcup_{i=1}^{\lambda-1} \mathcal{Q}_i$, we require that its initial length is bounded by $4d' \log^{10} N$.

At the beginning of the algorithm, we set $U_0 = \beta$, $U_1 = U_2 = \cdots = U_{\lambda-1} = \emptyset$, and $U_\lambda = V(H) \setminus \beta$. For all $1 \leq i < \lambda$, we also set $\mathcal{Q}_i = \emptyset$. It is easy to verity that this is a valid Layered Connectivity Data Structure. Throughout, we denote $\mathcal{Q} = \bigcup_{i=1}^{\lambda-1} \mathcal{Q}_i$. We also maintain, for every vertex $x \in \beta$ and layer $1 \leq i < \lambda$, the set $\mathcal{D}_i(x)$ of the descendants of x at layer i, and, for every vertex $v \in \bigcup_{i=1}^{\lambda-1} \mathcal{D}_i(x)$, a pointer from v to x. At the beginning of the algorithm, for all $x \in \beta$ and $1 \leq i < \lambda$, we set $\mathcal{D}_i(x) = \emptyset$. Additionally, for every layer $1 \leq i < \lambda$, for every vertex $v \in U^{<\lambda}$, if v serves as an endpoint of a path $K(u) \in \mathcal{Q}_i$, then we maintain a pointer from v to u.

For every edge $e \in E(H)$, we will maintain a list $S(e)$ of all vertices $v \in \bigcup_{i=1}^{\lambda-1} U_i$ with $e \in K(v)$. The task of maintaining these lists is quite straightforward. We will ignore it when we describe the remainder of the algorithm below, but we will show how to extend the algorithm in order to maintain these lists efficiently later.

For every vertex $v \in \bigcup_{i=1}^{\lambda-1} U_i$, we will maintain the length $\hat{\ell}(v)$ of the path $K(v) \in \mathcal{Q}$. When the path

$K(v)$ is first added to $\mathcal{Q}$, we initialize its length $\hat{\ell}(v)$. Then whenever a special edge on path $K(v)$ undergoes splitting, we increase $\hat{\ell}(v)$ by 1. The time that is required in order to maintain the values $\hat{\ell}(v)$ for vertices $v \in \bigcup_{i=1}^{\lambda-1} U_i$ is clearly subsumed by the time required to compute and to maintain the collection $\mathcal{Q}$ of paths. In our description of the remainder of the algorithm below, we will ignore the task of maintaining these values.

Before we provide the algorithm, we need one more definition, and a simple bound. We denote by G^0 the initial graph G. For every vertex $v \in V(G^0)$, we denote by $w^0(v)$ the weight of v at the beginning of the algorithm. If v lies in G at the end of the algorithm, then we let $w^*(v)$ be the weight of v at the end of the algorithm, and otherwise we let $w^*(v)$ be the weight of v just before it is deleted from G. We also denote $W^* = \sum_{v \in V(G^0)} w^*(v)$. Since the total increase in the lengths of edges over the course of the algorithm is at most $8n$, we get that $W^* \leq W^0 + 16n$. Since $N \geq \max\left\{n, W^0/8\right\}$, we get that $W^* \leq 32N$.

6.4.2 The Algorithm

We now describe the algorithm for executing a single phase. We denote by M the multiset of all vertices that ever lied in U_λ over the course of the algorithm. If a vertex is added to and removed from U_λ multiple times during the algorithm, we add multiple copies of that vertex to M. We will bound the running time of the algorithm in terms of $|M|$, and then later we will bound $|M|$. Whenever our algorithm computes a strongly well-structured $\frac{1}{d'}$-sparse cut (A, B) in G with $w(A) \geq 1.8|\beta \cap A|$, we call Procedure ProcessCut that we describe below. Following this procedure, the vertices that serve as endpoints of the edges in $E_G(A, B)$, along with the vertices of A (if $w(A) \leq w(B)$) or the vertices of B (otherwise) are deleted from G. We partition the execution of a single phase into three stages.

First stage. The first stage continues as long as the current graph G contains an edge of length greater than d'. Let $e = (u, v)$ be any such edge, and recall that it must be a special edge. Consider the cut (A, B) in G, where $A = \{u\}$, and $B = V(G) \setminus \{u\}$. It is immediate to verify that this cut is strongly well-structured. Moreover, since $w(u) = w(v) = \ell(e)$, while $|E_G(A, B)| = 1$, the sparsity of the cut (A, B) is at most $\frac{1}{d'}$, and $w(A) \geq 1.8|\beta \cap A|$ (as $|A| = 1$). We then output the cut (A, B), and call Procedure ProcessCut that is described below. This completes the description of the algorithm for the first stage of a phase. At the end of the first stage, graph G does not contain any edge whose length is greater than d'.

Second stage. The second stage is executed as long as $U_\lambda \neq \emptyset$. We let $1 \leq i < \lambda$ be the smallest index for which $\sum_{j=i+1}^{\lambda} |U_j| \geq 2^{\lambda-i-1}$ holds. Notice that, since $|U_\lambda| \geq 1$, if we choose $i = \lambda - 1$, then $\sum_{j=i+1}^{\lambda} |U_j| \geq 2^{\lambda-i-1} = 1$ holds, so such an index i must exist. We need the following claim.

Claim 6.8 $\sum_{j=i}^{\lambda} |U_j| \leq 2^{\lambda-i}$.

Proof: Assume otherwise, that is, $\sum_{j=i}^{\lambda} |U_j| > 2^{\lambda-i}$. Since we did not choose index $i-1$ instead of i, it must be the case that $i = 1$. Recall however that $\lambda = \lceil \log(64N) \rceil + 1$, and so $2^{\lambda-1} \geq 64N$. Therefore, $\sum_{j=1}^{\lambda} |U_j| \geq 64N$. However, $|V(H)| \leq W^* < 64N$ must hold, a contradiction. □

For every layer $i \leq j \leq \lambda - 1$ and vertex $v \in U_j$, we delete v from U_j and add it to U_λ. We also delete path $K(v)$ from set $\mathcal{Q}_j$, and delete v from the set $\mathcal{D}_j(x)$, where x is its ancestor-vertex in β. We denote by $k = |U_\lambda|$, so $2^{\lambda-i-1} \leq k \leq 2^{\lambda-i}$. Notice that the running time that we have spent so far in this iteration is $O(2^{\lambda-i}) + O(\lambda \cdot \log n) \leq O(k) + O(\log^2 N)$.

Next, we apply the algorithm for Procedure ReconnectLayer from Claim 6.1 to the current graphs G, H, and the index i. For brevity of notation, we denote this procedure by Π. We verify that we obtain a valid input to Procedure ReconnectLayer.

First, the current weight of all vertices in G is bounded by $W^* \leq 32N$, so $N \geq \max\{n, W/32\}$ holds. The inequalities $1 \leq r \leq \lceil\sqrt{\log N}\rceil$ and $1 \leq d' \leq \min\left\{\frac{2^{r\sqrt{\log N}}}{\log^8 N}, \frac{n}{8}\right\}$ hold from the definition of the ConnectToCenters problem. The length of every edge in G is at most d', and, for all $i \leq j < \lambda$, $U_j = \emptyset$ holds, while $|U_\lambda| \leq 2^{\lambda-i}$ from Claim 6.8. Recall that we have set $\rho = \frac{2^{12} \cdot W^0 \cdot \log^3 N}{|\beta^0|}$. Since $|\beta^0| \leq W^0$, we get that $\rho \geq \log^2 N$. Let $W = \sum_{v\in V(G)} w(v)$. From the problem definition, $W \leq W^* \leq 32W^0$, and $|\beta| \geq \frac{|\beta^0|}{4}$. Therefore:

$$\frac{W \cdot \log^3 N}{|\beta|} \leq \frac{128W^0 \cdot \log^3 N}{|\beta^0|} \leq \frac{\rho}{16},$$

as required.

It remains to verify that $|\beta| \geq \frac{W}{d' \log^2 N}$ holds. From the definition of the ConnectToCenters problem, $|\beta^0| \geq \frac{W^0}{256d'}$, and, throughout the algorithm, $|\beta| \geq \frac{|\beta^0|}{4} \geq \frac{W^0}{1024d'}$ holds. Since the total weight W of the vertices of G is bounded by $W^* \leq 32W^0$, and N is sufficiently large, we get that $|\beta| \geq \frac{W^0}{1024d'} \geq \frac{W}{2^{15}d'} \geq \frac{W}{d' \log^2 N}$ holds. Therefore, we obtain a valid input for Procedure ReconnectLayer. Recall that the running time of procedure Π is bounded by:

$$O\left(c_1 \cdot W \cdot k \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+14}\right)$$
$$\leq O\left(c_1 \cdot W^0 \cdot k \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+14}\right),$$

and the probability that it returns "FAIL" is at most $\frac{1}{N^{10}}$. If the procedure returns "FAIL", then we terminate our algorithm, and return "FAIL" as well. Otherwise, we consider three cases.

In the first case, the procedure returns a weakly well-structured cut (A', B') in H with $|A'| \geq \frac{511|V(H)|}{512}$, $|A' \cap \beta| < \frac{2|\beta|}{\log^3 N}$, and $|E_H(A', B')| \leq \frac{|V(H)|}{1024d' \cdot \log^6 N}$. In this case, we return the cut (A', B'), and the algorithm terminates. We say that Procedure Π produced a type-1 outcome in this case.

In the second case, the procedure returns a valid Layered Connectivity Data Structure, in which, for all $i < i' < \lambda$, $U_{i'} = \emptyset$ and $|U_\lambda| \leq \frac{k}{4\lambda}$ holds; additionally, layers $U_1, \ldots, U_{i-1}$, together with the corresponding collections of paths $\mathcal{Q}_1, \ldots, \mathcal{Q}_{i-1}$ remain the same as in the input Layered Connectivity Data Structure. In this case we say that Procedure Π produced a type-2 outcome. Notice that, at the beginning of the procedure, $|U_\lambda| = k$ held, while at the end of the procedure, $|U_\lambda| \leq \frac{k}{4\lambda}$. Therefore, at least $k\left(1 - \frac{1}{4\lambda}\right)$ vertices were removed from U_λ by this procedure. For each such vertex v, we *charge* the vertex $O\left(c_1 \cdot W^0 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+14}\right)$ units for the execution of the procedure, so that the total charge to all vertices that were removed from U_λ by the procedure is at least the total running time of the procedure.

Lastly, we consider the third case, where Procedure Π returned a strongly well-structured $\frac{1}{4d'}$-sparse cut (A, B) in G with $w(A), w(B) \geq \frac{k}{2^{14}\log N}$ and $w(A) \geq 1.8|\beta \cap A|$. In this case, we say that the outcome of Procedure Π is of type 3. We then output the cut (A, B), and call Procedure ProcessCut that is described below. Recall that, following this cut, all vertices that serve as endpoints of the edges in $E_G(A, B)$, and also either all vertices of A or all vertices of B, are deleted from G. Let Z denote the set of all vertices that are deleted from G. Notice $w(Z) \geq \min\{w(A), w(B)\} \geq \frac{k}{2^{14}\log N}$. For every vertex $v \in Z$, we charge the vertex $O\left(c_1 \cdot W^0 \cdot w(v) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+15}\right)$ units for the execution of the iteration, so that the total charge to all vertices of Z is at least as high as the running time of the iteration.

Third Stage. The second stage terminates once we obtain a valid Layered Connectivity Data Structure (with relaxed Requirement L1), with $U_\lambda = \emptyset$. Recall that, following this, the algorithm is given a set $E' \subseteq E(G)$ of special edges whose lengths need to be doubled, and a set $\hat{\beta} \subseteq \beta$ of vertices, that need to be deleted from β. For every special edge $e \in E'$, we consider each of the special edges $e' \in P^*(e)$ one by one. We start by performing a splitting update of the edge e', that introduces two new special edges e_1, e_2 in graph H, that replace edge e', and two new vertices, that we add to set U_λ. We initialize $S(e'_1) = S(e'_2) = \emptyset$. For every vertex $v \in S(e)$, we update the corresponding path $K(v)$ by splitting the edge e', and we update the length $\hat{\ell}(v)$ of the path by increasing it by (additive) 2. If $\hat{\ell}(v)$ becomes greater than $16d' \log^{10} N$, then let U_j be the layer to which v belongs. We delete v from U_j and add it to U_λ. We also delete the path $K(v)$ from $\mathcal{Q}_j$, and we say that path $K(v)$ is *destroyed*. We also delete v from set $\mathcal{D}_j(x)$, where $x \in \beta$ is the ancestor of v. Let Λ' denote the total initial length of the edges in E', so the lengths of all edges increased by additive Λ' in this iteration, and the number of edge-splitting operations performed in graph H is bounded by Λ'. Since, from Requirement L1, for all $1 \leq i < \lambda$, the paths in $\mathcal{Q}_i$ cause edge-congestion at most $4d' \log^9 N$, and since $\lambda \leq O(\log N)$, we get that, for every edge $e' \in E(H)$, $|S(e)| \leq O(d' \log^{10} N)$. Therefore, the total running time of the algorithm for Stage 3 so far is $O(\Lambda' \cdot d' \cdot \log^{10} N)$.

Next, we process the vertices of $\hat{\beta}$, that are deleted from set β. We consider the vertices $x \in \hat{\beta}$ one by one. For each such vertex x, for every layer $1 \leq j < \lambda$, and for every descendant $v \in \mathcal{D}_i(x)$ of x at layer j, we delete v from U_j and add it to U_λ. We also delete the path $K(v)$ from $\mathcal{Q}_j$ (we may sometimes say that the path is *destroyed*). This completes the description of the algorithm for the third stage of a phase. Recall that, from Property L2 of the Layered Connectivity Data Structure, for every vertex $x \in \beta$, for every layer $1 \leq j \leq \lambda - 1$, $|\mathcal{D}_j(x)| \leq \rho$. Therefore, the total number of vertices that are added to U_λ due to the deletion of their ancestors from β is bounded by $O(|\hat{\beta}| \cdot \rho \cdot \lambda) \leq O(|\hat{\beta}| \cdot \rho \cdot \log N)$. The running time of Stage 3 is bounded by $O(\Lambda' \cdot d' \cdot \log^{10} N) + O(|\hat{\beta}| \cdot \rho \cdot \log N) \leq O(\Lambda' \cdot W^0 \cdot \log^{10} N) + O(|\hat{\beta}| \cdot \rho \cdot \log N)$, since $d' \leq n \leq W^0$ from the problem definition. This completes the description of the algorithm for a single phase.

Recall that Λ bounds the total increase in the lengths of edges of the course of the algorithm. Let $\hat{W}$ denote the total weight of all vertices that were ever deleted from G over the course of the algorithm, and let β' be the set of all vertices that were deleted from β over the course of the algorithm. Observe that the vertices of $\beta^0 \setminus \beta'$ remain in G throughout the algorithm, and so:

$$\hat{W} \leq W^* - |\beta^0 \setminus \beta'| \leq W^0 + 2\Lambda - |\beta^0| + |\beta'|. \tag{12}$$

From the above discussion, the total running time of algorithms for stages 1, 2 and 3 (excluding the running times of the executions of Procedure Π and Procedure ProcessCut), over the course of the entire algorithm, is bounded by:

$$O(|\beta'| \cdot \rho \cdot \log N) + O(\Lambda \cdot d' \cdot \log^{10} N).$$

Since, from the problem definition, $d' \leq n \leq W^0$, this is bounded by:

$$O(|\beta'| \cdot \rho \cdot \log N) + O(\Lambda \cdot W^0 \cdot \log^{10} N).$$

Next, we bound the total running time of all executions of Procedure ReconnectLayer over the course of the algorithm. Observe first that type-1 outcome of the procedure may only occur once over the course of the algorithm. The running time of Procedure ReconnectLayer in case of type-1 outcome is bounded by:

$$O\left(c_1 \cdot W^0 \cdot k \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+14}\right),$$

where $k \leq |V(H) \setminus \beta|$. Recall that β' is the set of all vertices that were deleted from β over the course of the algorithm, so $|\beta| \geq |\beta^0| - |\beta'|$. Recall also that $|V(H)| \leq W^* \leq W^0 + 2\Lambda$. Therefore, $k \leq |V(H) \setminus \beta| \leq W^* - |\beta^0| + |\beta'| \leq W^0 + 2\Lambda - |\beta^0| + |\beta'|$. We then get that the total running time of Procedure ReconnectLayer when type-1 outcome occurs is bounded by:

$$O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + \Lambda + |\beta'|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+14}\right).$$

Recall that we have charged the running times of the executions of Procedure Π in which type-2 or type-3 outcomes occured to vertices that were removed from U_λ by the procedure (in case of type-2 outcome), or to the weights of the vertices that were deleted from G following the procedure. Recall also that M is the multiset of all vertices that ever lied in U_λ. The total running time of the remaining executions of Procedure Π is bounded by:

$$\begin{aligned}
&O\left(c_1 \cdot W^0 \cdot (\hat{W} + |M|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+15}\right)\\
&\qquad\leq O\left(c_1 \cdot W^0 \cdot (W^0 + |\beta'| - |\beta^0| + \Lambda + |M|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+15}\right),
\end{aligned}$$

from Inequality 12. The total running time of the algorithm, excluding Procedure ProcessCut, is therefore bounded by:

$$O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda + |M|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+15}\right).$$

Next, we describe Procedure ProcessCut and analyze its running time, and then bound $|M|$.

6.4.3 Procedure ProcessCut

Procedure ProcessCut is called whenever our algorithm produces a strongly well-structured $\frac{1}{d'}$-sparse cut (A, B) in G with $w(A) \geq 1.8|\beta \cap A|$. Let Z be the set of vertices of G that contains all endpoints of the edges in $E_G(A, B)$. Additionally, if $w(A) \leq w(B)$, we add the vertices of A to Z, and we add the vertices of B to Z otherwise. Recall that, following this cut, the vertices of Z are deleted from G. Additionally, some vertices may be deleted from β. We denote the latter set of vertices by $\hat{\beta}$, and we assume that $Z \cap \beta \subseteq \hat{\beta}$.

We start by updating the graph H, as follows: we delete the vertices of Z from H, and, for every special edge $e \in E(G)$ that is incident to a vertex of Z, we delete all vertices and edges on the corresponding path $P^*(e)$ from H. For every vertex v that is deleted from H, if U_j is the layer of the Layered Connectivity Data Structure to which v belongs, we delete v from U_j and from set $\mathcal{D}_j(x)$, where $x \in \beta$ is the ancestor of v, and we also delete path $K(v)$ from $\mathcal{Q}_j$. Note that the update of graph H takes time $O(w(Z))$. We denote by Z' the set of vertices that were deleted from H, so $|Z'| \leq O(w(Z))$.

Next, we consider the vertices $x \in \hat{\beta}$ one by one. For each such vertex x, for every layer $1 \leq j < \lambda$, and for every descendant $v \in \mathcal{D}_j(x)$ of x at layer j, we delete v from U_j and add it to U_λ. We also delete the path $K(v)$ from $\mathcal{Q}_j$ (we may sometimes say that the path is *destroyed*). Additionally, if x remains in H, we add it to U_λ as well. Let n' be the number of vertices that were added to U_λ so far. Recall that, from Property L2 of the Layered Connectivity Data Structure, for every vertex $x \in \hat{\beta}$, for every layer $1 \leq j \leq \lambda - 1$, $|\mathcal{D}_j(x)| \leq \rho$. Therefore, $n' \leq |\hat{\beta}| \cdot \lambda \cdot \rho \leq 8|\hat{\beta}| \cdot \rho \cdot \log N$. The running time of the algorithm so far is bounded by $O(w(Z) + |\hat{\beta}| \cdot \rho \cdot \log N)$.

Lastly, it is possible that, for some vertices $v \in \bigcup_{j=1}^{\lambda-1} U_j$, some vertex of $K(v)$ was deleted from H by the procedure, while vertex v itself was not deleted. Note that, if the ancestor $x \in \beta$ of v was deleted from β, then we have already removed v from its corresponding layer. Let $V' \subseteq \bigcup_{j=1}^{\lambda-1} U_j$ denote the set of all vertices $v \in \bigcup_{j=1}^{\lambda-1} U_j$, such that $v \notin Z'$, the ancestor $x \in \beta$ of v is not in $\hat{\beta}$, and some vertex on path $K(v)$ is in Z'. We further partition set V' into two subsets: set V'_1 contains all vertices $v \in V'$, such that the last vertex u on path $K(v)$ lies in Z'; while set V'_2 contains all remaining vertices.

From Requirement L3 of Layered Connectivity Data Structure, $|V'_1| \leq \lambda \cdot Z' \leq O(w(Z) \cdot \log N)$. Since the Layered Connectivity Data Structure data structure maintains, for every vertex $v \in V'$ a pointer from the last endpoint u of $K(v)$ to vertex v itself, set V'_1 can be computed in time $O(|V'_1|) \leq O(w(Z) \cdot \log N)$.

In order to compute the set V'_2 of vertices, consider the collection $E_G(A, B)$ of edges. Recall that, for every edge $e \in E_G(A, B)$, we have defined a path $P^*(e)$ that represents edge e in graph H'. Let $E' \subseteq E(H)$ be the set of edges that contains, for each edge $e \in E_G(A, B)$, the first and the last edge on path $P^*(e)$. We claim that, for every vertex $v \in V'_2$, path $K(v)$ must contain an edge of E'.

Observation 6.9 *For every vertex $v \in V'_2$, path $K(v)$ must contain an edge of E'.*

Proof: Assume first that $w(A) \leq w(B)$. Consider the graph $H \setminus E'$, and let (a, b) be any edge in this graph with $a \in Z'$ and $b \notin Z'$. From the definition of the set Z' of vertices and the set E' of edges, it must be the case that a is the last endpoint of some path $P^*(e)$ for an edge $e \in E_G(A, B)$; equivalently, $a \in V(G)$ is the tail of one of the edges in $E_G(A, B)$. But then $a \in L'$, and it only has a single incoming edge in H, which must belong to E'. Consider now some vertex $v \in V'_2$, and its corresponding path $K(v)$. Assume for contradiction that $K(v)$ does not contain an edge of E'. Since $K(v)$ connects two vertices of $V(H) \setminus Z'$, and contains at least one vertex of Z' as an inner vertex, it must contain an edge (a, b) with $a \in Z'$ and $b \notin Z'$. But the only edge entering a lies in E', a contradiction.

If $w(A) > w(B)$, then the proof is the same, except that we consider an edge (a, b) with $a \notin Z'$ and $b \in Z'$. In this case, b must be a head of an edge of $E_G(A, B)$, and so it has no outgoing edges in $H \setminus E'$. □

In order to compute the set V'_2 of vertices, we consider the edges $e \in E_G(A, B)$ one by one. When edge e is considered, we let e' and e'' be the first and the last edges on the corresponding path $P^*(e)$. We then add to V'_2 all vertices that lie in set $S(e') \cup S(e'')$. Notice that, from Requirement L1 of the Layered Connectivity Data Structure, the congestion caused by the set $\mathcal{Q}$ of paths is bounded by $O(d'\lambda \log^9 N) \leq O(d' \log^{10} N)$, and so for every edge $e^* \in E(H)$, $|S(e^*)| \leq O(d' \log^{10} N)$. Altogether, we get that: $|V'_2| \leq O\left(|E_G(A, B)| \cdot d' \cdot \log^{10} N\right)$. Since cut (A, B) is $\frac{1}{d'}$-sparse, we get that $|E_G(A, B)| \leq \frac{\min\{w(A), w(B)\}}{d'} \leq \frac{w(Z)}{d'}$. Overall, we get that $|V'_2| \leq O(w(Z) \cdot \log^{10} N)$, and set V'_2 can be computed in time $O(w(Z) \cdot \log^{10} N)$. From our discussion, $|V'| \leq O(w(Z) \cdot \log^{10} N)$, and it can be computed in time $O(w(Z) \cdot \log^{10} N)$.

For every vertex $v \in V'$, let $1 \leq j \leq \lambda$ be the index of the layer to which v belongs. If $j < \lambda$, then we delete v from U_j and add it to U_λ, and we also delete the path $K(v)$ from $\mathcal{Q}_j$. Additionally, if $x \in \beta$ is the ancestor of v, then we delete v from set $\mathcal{D}_j(x)$.

From our discussion, the total number of vertices that were added to U_λ over the course of the procedure is bounded by $O(w(Z) \cdot \log^{10} N) + O(|\hat{\beta}| \cdot \rho \cdot \log N)$, and the running time of the procedure is bounded by $O(w(Z) \cdot \log^{10} N) + O(|\hat{\beta}| \cdot \rho \cdot \log N)$.

Recall that we have denoted by $\hat{W}$ the total weight of all vertices that were ever deleted from G over the course of the algorithm, and by β' the set of all vertices that remain in β at the end of the algorithm. Recall also that, from Inequality 12, $\hat{W} \le W^0 + 2\Lambda - |\beta^0| + |\beta'|$. Therefore, the total number of vertices that were ever added to set U_λ by Procedure ProcessCut is bounded by:

$$O(\hat{W} \cdot \log^{10} N) + O(|\beta'| \cdot \rho \cdot \log N) \le O\left((W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N\right),$$

and the total running time of all calls to Procedure ProcessCut over the course of the algorithm is bounded by

$$O(\hat{W} \cdot \log^{10} N) + O(|\beta'| \cdot \rho \cdot \log N) \le O\left((W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N\right).$$

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

$$O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda + |M|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+15}\right). \tag{13}$$

6.4.4 Bounding $|M|$ and the Running Time of the Algorithm

Recall that M is the multiset containing all vertices that were ever added to U_λ over the course of the phase. We partition the set M into six subsets, $M_1, \ldots, M_6$, depending on the process through which the vertices were added to U_λ, and we then bound the cardinality of each set separately.

Set M_1. Recall that at the beginning of the algorithm, every vertex of $V(H) \setminus \beta^0$ is added to U_λ. Set M_1 contains all these vertices, and, additionally, whenever a new vertex is inserted into graph H due to the splitting of a special edge, this new vertex, which is then added to U_λ, is also added to M_1. Lastly, if Procedure ReconnectLayer ever terminates with a type-1 outcome, then all vertices that were added to U_λ in the same iteration, just prior to that execution of Procedure ReconnectLayer, are also added to set M_1. Since the vertices of $\beta^0 \setminus \beta'$ always lie in U_0 and are never added to U_λ, it is easy to see that $|M_1|$ is asymptotically bounded by the total number of vertices ever present in H, excluding the vertices of $|\beta^0 \setminus \beta'|$. Therefore, $|M_1| \le O(W^* - |\beta^0 \setminus \beta'|) \le O(W^0 + 2\Lambda - |\beta^0| + |\beta'|)$.

Set M_2. The second set M_2 contains all vertices v that were added to U_λ when the length of the current path $K(v) \in \mathcal{Q}$ exceeded $16d' \log^{10} N$. Recall that, when path $K(v)$ was added to $\mathcal{Q}$, its length was bounded by $4d' \log^{10} N$. Therefore, between the time path $K(v)$ was added to $\mathcal{Q}$, and the time it was destroyed, its edges underwent at least $4d' \log^{10} N$ splitting updates. Recall that the total increase in the lengths of all edges in G is denoted by Λ, so the total number of edge splitting updates in graph H is bounded by Λ. Since every edge of H may belong to at most $4d'\lambda \log^9 N \le 32d' \log^{10} N$ paths in $\mathcal{Q}$, we get that the total number of edge splitting updates that the paths in $\mathcal{Q}$ undergo over the course of the algorithm is bounded by $O(\Lambda \cdot d' \log^{10} N)$. Since each destroyed path must undergo at least $4d' \log^{10} N$ edge-splitting updates, we get that $|M_2| \le O(\Lambda)$.

Set M_3. The third set, M_3, contains all vertices v that were added to U_λ because the ancestor-vertex of v was deleted from β during Phase 3 of the algorithm. Note that, from Property L2 of the

Layered Connectivity Data Structure, every vertex $x \in \beta$ has at most $\lambda \cdot \rho$ descendants at any time. Therefore, $|M_3| \leq O(|\beta'| \cdot \rho \cdot \lambda) \leq O(|\beta'| \cdot \rho \cdot \log N)$.

Set M_4. The fourth set, M_4, contains all vertices that were added to U_λ by Procedure ProcessCut. As we showed already, $|M_4| \leq O\left((W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N\right)$.

Set M_5. The fifth set, M_5, contains all vertices that were added to U_λ during the iterations of Stage 2, just prior to the call to Procedure ReconnectLayer, that resulted in a type-3 outocme of the procedure. Recall that in each such iteration, the number of vertices that were added to U_λ was bounded by k, while the total weight of the vertices that are subsequently deleted from G is at least $\Omega\left(\frac{k}{\log N}\right)$. Therefore, $|M_5| \leq O(\hat{W} \cdot \log N) \leq O\left((W^0 + 2\Lambda - |\beta^0| + |\beta'|) \cdot \log N\right)$ from Inequality 12,

So far we have defined five subsets $M_1, \ldots, M_5$ of the set M, and we have shown that:

$$\sum_{j=1}^{5} |M_j| \leq O\left((W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N\right). \tag{14}$$

Set M_6. We let the sixth set, M_6, contain all remaining vertices of M. Each such vertex v must have been added to U_λ during an iteration of the second stage, just prior to a call to Procedure ReconnectLayer, that resulted in a type-2 outcome. Consider any such iteration of Phase 2, and let $1 \leq i < \lambda$ be the layer that was chosen by the iteration. We then say that this iteration of Phase 2 is a *type-i iteration*. For brevity, in the remainder of the proof, we use "type-i iteration" only to refer to type-i iterations of Phase 2 in which Procedure ReconnectLayer resulted in a type-2 outcome. Recall that, at the beginning of such an iteration, $\sum_{j=i+1}^{\lambda} |U_j| \geq 2^{\lambda-i-1}$ holds, and the vertices of $\bigcup_{j=i}^{\lambda-1} U_j$ are then moved from their current layers to U_λ. Let $k = \sum_{j=i}^{\lambda} |U_j|$, so $2^{\lambda-i-1} \leq k \leq 2^{\lambda-i}$. Then at the beginning of the iteration, at most k vertices were added to U_λ, while at the end of the iteration, $|U_\lambda| \leq \frac{k}{4\lambda}$ held. Moreover, layers $U_1, \ldots, U_{i-1}$ remain unchanged over the course of the iteration.

From the description of our algorithm, for any pair $1 \leq i', i'' \leq \lambda$ of indices, a vertex may only move directly from layer $U_{i'}$ to layer $U_{i''}$ if either $i' = \lambda$ or $i'' = \lambda$. At the beginning of a type-i iteration, $|U^{>i}| \geq 2^{\lambda-i-1}$ holds, while at the end of the iteraiton, $|U^{>i}| \leq \frac{k}{4\lambda} \leq \frac{2^{\lambda-i}}{4\lambda}$. Therefore, between every pair of type-i iterations, at least $2^{\lambda-i-1} - \frac{2^{\lambda-i}}{4\lambda} \geq 2^{\lambda-i-2}$ vertices must have moved from $\bigcup_{i'=1}^{i} U_{i'}$ to U_λ. For all $1 \leq i < \lambda$, we denote by $M_6^i \subseteq M_6$ the set of vertices that were added to M_6 during type-i iterations. We start by bounding the number of type-i iterations, for $1 \leq i < \lambda$.

Claim 6.10 *There is a large enough constant c^*, such that for all $1 \leq i < \lambda$, the total number of type-i iterations is bounded by $\frac{c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N}{2^{\lambda-i}}$.*

Proof: The proof is by induction. The base case is when $i = 1$. From the above discussion, between every pair of type-1 iterations, at least $2^{\lambda-3}$ new vertices must have been moved from U_1 to U_λ. All such vertices must belong to $M_1 \cup \cdots \cup M_5$. Since $\sum_{j=1}^{5} |M_j| \leq O\left((W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N\right)$, and c^* is a sufficiently large constant, we get that the total number of type-1 iterations is bounded by:

$$O\left(\frac{(W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N}{2^{\lambda}}\right) \leq \frac{c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N}{2^{\lambda-1}}.$$

Consider an index $1 < i < \lambda$, and assume that the claim holds for all indices $1 \leq i' < i$. We define a potential function $\Phi_i = \sum_{j=i+1}^{\lambda} |U_j|$. Recall that, at the beginning of a type-i iteration,

$\Phi_i = \sum_{j=i+1}^{\lambda} |U_j| \geq 2^{\lambda-i-1}$ holds, while at the end of the iteration, $|U_\lambda| \leq \frac{2^{\lambda-i}}{4\lambda}$, and for all $i < i' < \lambda$, $U_{i'} = \emptyset$, so $\Phi_i \leq \frac{2^{\lambda-i}}{4\lambda}$. Therefore, in every type-i iteration, Φ_i decreases by at least $2^{\lambda-i-1} \cdot \left(1 - \frac{1}{2\lambda}\right) \geq 0.48 \cdot 2^{\lambda-i}$. The potential Φ_i is increased whenever vertices are added to sets $M_1, \ldots, M_5$, and the total increase due to the addition of such vertices to $M_1, \ldots, M_5$ is bounded by $\sum_{j=1}^{5} |M_j|$. Additionally, for all $1 \leq i' < i$, each type-i' iteration may increase the potential Φ_i by at most $\frac{2^{\lambda-i'}}{4\lambda}$. Since, from the induction hypothesis, the number of type-i' iterations is bounded by $\frac{c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N}{2^{\lambda-i'}}$, the total increase in the potential Φ_i due to type-i' iterations is bounded by:

$$\frac{c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N}{2^{\lambda-i'}} \cdot \frac{2^{\lambda-i'}}{4\lambda} \leq \frac{c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10}}{4\lambda}.$$

Therefore, the total increase in the potential Φ_i over the course of the entire algorithm is bounded by:

$$\begin{aligned}\sum_{j=1}^{5} |M_j| + (i-1) \cdot & \frac{c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N}{4\lambda} \\ & \leq 0.26 c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N.\end{aligned}$$

Since every successful type-i iteration decreases the potential Φ_i by at least $0.48 \cdot 2^{\lambda-i}$, we get that the total number of type-i iterations is bounded by:

$$\frac{0.26 c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N}{0.48 \cdot 2^{\lambda-i}} \leq \frac{c^* \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N}{2^{\lambda-i}}.$$

□

Since, for all $1 \leq i < \lambda$, every type-i iteration adds at most $O(2^{\lambda-i})$ vertices to U_λ, we get that $|M_6^i| \leq O\left((W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{10} N\right)$, and altogether:

$$|M_6| = \sum_{i=1}^{\lambda-1} |M_6^i| \leq O\left((W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \lambda \cdot \log^{10} N\right).$$

Combining this with Inequality 14, and substituting $\lambda \leq O(\log N)$, we get that:

$$|M| \leq O\left((W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot \log^{11} N\right).$$

Lastly, substituting this bound on $|M|$ into the bound from Equation 13, we get that the total running time of the algorithm so far is bounded by:

$$\begin{aligned} & O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda + |M|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+15}\right) \\ & \quad \leq O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+26}\right)\end{aligned}$$

6.4.5 Maintaining the Additional Data Structures

It remains to show an algorithm that maintains, for every edge $e \in E(H)$, the set $S(e)$ of vertices, whose corresponding path $K(v)$ contains e. Additionally, for every vertex $x \in \beta$, we need to maintain, for all $1 \le i < \lambda$, the set $\mathcal{D}_i(x)$ of the descendants of x at layer i, and, for every vertex $v \in \bigcup_{i=1}^{\lambda-1} U_i$, a pointer to its unique ancestor in β. Lastly, for every layer $1 \le i \le \lambda - 1$ and vertex $v \in \bigcup_{j=1}^{i-1} U_j$, if v serves as an endpoint of a path $K(u) \in \mathcal{Q}_i$, then we need to maintain a pointer from v to u. We now show how to extend our algorithm, in a straightforward manner, to maintain these additional data structures.

At the beginning of the algorithm, for every edge $e \in E(H)$, we set $S(e) = \emptyset$, and for every vertex $x \in \beta$ and index $1 \le i < \lambda$, we set $\mathcal{D}_i(x) = \emptyset$. Whenever a new vertex v is added to any layer U_i, for $1 \le i < \lambda$, we inspect its corresponding path $K(v)$, that is added to $\mathcal{Q}_i$. For every edge $e \in E(K(v))$, we add vertex v to the list $S(e)$. Let y denote the last endpoint of path $K(v)$, and let $x \in \beta$ be the unique ancestor of y. We then add v to $\mathcal{D}_i(x)$, and we add a pointer from v to x, and another pointer from y to v.

Whenever any vertex u is deleted from a set U_i, for any $1 \le i < \lambda$, we again inspect the corresponding path $K(v)$, and, for every edge $e \in E(K(v))$, we delete v from set $S(v)$. If y is the other endpoint of path $K(v)$, then we delete the pointer from y to v. Additionally, if $x \in \beta$ is the unique ancestor of v at level 0, then we delete v from $\mathcal{D}_i(x)$. Note that the time required to execute all these updates is asymptotically bounded by the time required to compute all paths $K(v)$ that are ever added to $\mathcal{Q}$.

Therefore, the total update time of the algorithm remains bounded by:

$$O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + |\beta'| \cdot \rho + \Lambda) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+26}\right).$$

Recall that the algorithm from Procedure ReconnectLayer may terminate with a "FAIL", and if this happens, our algorithm terminates with a "FAIL" as well. However, the number of calls to Procedure ReconnectLayer is bounded by $|M| \le \tilde{O}((W^0)^2) \le N^3$, and the probability that a single call terminates with a "FAIL" is bounded by $\frac{1}{N^{10}}$. Therefore, overall, the probability that our algorithm terminates with a "FAIL" is bounded by $\frac{1}{N^7}$.

6.5 An Algorithm for the MaintainSpecialCluster Problem

In this subsection we prove the second part of Theorem 4.12, by showing that an algorithm for the r-restricted RouteAndCut problem implies an algorithm for the r-restricted MaintainSpecialCluster problem with the required running time. We summarize this result in the following theorem.

Theorem 6.11 *Suppose that, for some parameter $r \ge 1$, there exists a randomized algorithm for the r-restricted* RouteAndCut *problem, that, given as an input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most $c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2}$. Then there exists a randomized algorithm for the r-restricted* MaintainSpecialCluster *problem, that, on input $(G, \beta, \Delta, \eta, d, N)$ has running time at most: $c_1 \cdot W^0(G) \cdot (W^0(G) - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r}$.*

The remainder of this subsection is dedicated to the proof of Theorem 6.11. Recall that we are given as input a perfect well-structured graph $G = (L, R, E)$ with a proper assignment of lengths $\ell(e)$ to its edges $e \in E$, that are given in the adjacency-list representation, such that the length of every special edge is 1. We are also given a subset $\beta \subseteq V(G)$ of vertices with $|\beta| \ge \frac{99|V(G)|}{100}$, and parameters $N \ge W^0(G)$, $1 \le \Delta \le \frac{|\beta|}{(\log N)^{128}}$, $1 \le \eta \le \Delta$, and $d \ge (\log N)^{64}$, such that $\frac{\Delta \cdot d}{\eta} \le |V(G)| - |\beta| + \Delta \cdot (\log N)^{64}$. Lastly, we are given an integer $1 \le r \le \left\lceil \sqrt{\log N} \right\rceil$, and we are guaranteed that $d \le 2^{r \cdot \sqrt{\log N}}$.

Over the course of the algorithm, vertices may be deleted from set β, and the lengths of some edges may be increased, but the length of every special edge incident to a vertex that currently lies in β

is always 1. We denote by β^0 the initial set β of vertices. Once $|\beta|$ falls below $\frac{|\beta^0|}{2}$, the algorithm terminates.

Recall that the algorithm consists of at most Δ iterations. At the beginning of every iteration i, the algorithm is given a vertex $x \in V(G)$, and it must return a simple path P_i in G connecting x to any vertex $y \in \beta$, such that the length of the path is at most d, and no inner vertices on the path lie in β (if $x \in \beta$, that we require that path P_i only consists of the vertex x). After that, the length of some special edges on path P_i may be doubled, and vertex y is deleted from β. In this case, we say that vertex y was deleted from β *directly.* If $\tau' > \tau$ are two times at which the length of some edge $e \in E(G)$ is doubled, then we are guaranteed that the algorithm returned at least η paths in response to queries during the time interval $(\tau, \tau']$ that contained e.

The algorithm may, at any time, produce a strongly well-structured cut (X, Y) in G of sparsity $\Phi_G(X, Y) \le \frac{(\log N)^{64}}{d}$, such that $w(X) \ge 1.8|\beta \cap X|$ holds. Let $J' \subseteq Y$ denote the set of vertices that serve as endpoints of the edges of $E_G(X, Y)$. The vertices of $X \cup J'$ are then deleted from G, and the algorithm continues with the resulting graph $G = G[Y \setminus J']$. Let Z be the set of vertices that are deleted from G following this cut. Notice that the vertices of $Z \cap \beta$ are deleted from graph G, and hence from set β as well. We say that the vertices of $Z \cap \beta$ are deleted from β *indirectly.*

Let Λ denote the total increase in the lengths of edges over the course of the algorithm, and denote $W^0 = W^0(G) = n$. We also denote the initial graph G by G^0, and the set of all vertices that were deleted from β over the course of the algorithm, either directly or indirectly, by β'. We need the following claim and its corollary, whose proofs are deferred to Section C.3 and C.4 of Appendix, respectively.

Claim 6.12 $\Lambda \le 4n - 4|\beta^0| + 5\Delta \cdot (\log N)^{64} \le 4n$ *and* $|\beta'| \le 45n - 45|\beta^0| + 56\Delta \cdot (\log N)^{64}$.

Corollary 6.13 *Throughout the algorithm,* $\sum_{v \in V(G) \setminus \beta} w(v) \le 0.2|\beta|$.

Note that the subdivided graph H of G is identical to graph G, so we can use the adjacency list representation of G also as the adjacency list representation of H. We define a parameter $d' = \frac{d}{(\log N)^{64}}$. Recall that, from the definition of the MaintainSpecialCluster problem, we are guaranteed that $d \le 2^{r \cdot \sqrt{\log N}}$, and also that $d \le \frac{\eta}{\Delta}\left(n - |\beta^0| + \Delta \cdot (\log N)^{64}\right) \le n$, since $\eta \le \Delta$, and $\Delta \le \frac{|\beta^0|}{(\log N)^{128}}$ from the problem definition. Therefore, $d' \le \min\left\{\frac{2^{r\sqrt{\log N}}}{\log^8 N}, \frac{n}{8}\right\}$. Since $|\beta^0| \ge \frac{99n}{100}$ from the problem definition, and $d' \ge 1$ (as $d \ge (\log N)^{64}$), we get that $|\beta^0| \ge \frac{W^0}{256d'}$.

We apply the algorithm for the ConnectToCenters problem from Theorem 6.7 to graph G, set β of vertices, and parameters N, d', and r. From our discussion, this is a valid input to the problem. Recall that the algorithm uses a parameter $\rho = \frac{2^{12} \cdot W^0 \cdot \log^3 N}{|\beta^0|}$. Since, from the definition of the MaintainSpecialCluster problem, $|\beta^0| \ge \frac{99n}{100}$, while $W^0 = n$, we get that $\rho = O(\log^3 N)$. Our algorithm consists of at most Δ phases, that correspond to the phases of the ConnectToCenters problem. Recall that the purpose of each such phase is to construct a valid Layered Connectivity Data Structure with $U_\lambda = \emptyset$. Our algorithm will only respond to queries at the end of each phase, once such a data structure is obtained.

We now consider a single phase of the algorithm for the ConnectToCenters problem, and one of its iterations. Assume first that the algorithm produces a strongly well-structured $\frac{1}{d'}$-sparse cut (A, B) in G with $w(A) \ge 1.8|\beta \cap A|$. Since $d' = \frac{d}{(\log N)^{64}}$, the sparsity of the cut is at most $\frac{(\log N)^{64}}{d}$. We claim that $w(A) < w(B)$ must hold in the following simple obsevation, whose proof is deferred to Section C.5 of appendix.

Observation 6.14 *$w(A) < w(B)$ must hold.*

We delete from G the vertices of A, and the endpoints of the edges of $E_G(A, B)$ that lie in B, and continue with the resulting graph.

Next, we show that the algorithm to the ConnectToCenters may never produce a weakly well-structured cut (A', B') in graph H, with $|A'| \geq \frac{511|V(H)|}{512}$, $|A' \cap \beta| < \frac{2|\beta|}{\log^3 N}$, and $|E_H(A', B')| \leq \frac{|V(H)|}{d' \cdot \log^6 N}$. Indeed, assume for contradiction that the algorithm produces such a cut. Since every special edge incident to a vertex of β has length 1, we get that $|V(H) \setminus \beta| = \sum_{v \in V(G) \setminus \beta} w(v)| \leq 0.2|\beta|$ from Corollary 6.13. But then:

$$|A'| \leq |V(H) \setminus \beta| + \frac{2|\beta|}{\log^3 N} \leq 0.21|\beta|.$$

Since $|V(H)| \geq |\beta|$, it is then impossible that $|A'| \geq \frac{511|V(H)|}{512}$, a contradiction.

Therefore, if the algorithm for the ConnectToCenters problem does not produce a a strongly well-structured $\frac{1}{d'}$-sparse cut (A, B) in G with $w(A) \geq 1.8|\beta \cap A|$, it must produce a valid Layered Connectivity Data Structure, with $U_\lambda = \emptyset$. In this case, our algorithm is ready to respond to a query. Let x be a query vertex that it receives. If $x \in \beta$, then we return a path consisting of the vertex x itself. Otherwise, we construct a path P connecting x to some vertex $y \in \beta$, as follows. Since $U_\lambda = \emptyset$, there must be an integer $1 \leq i < \lambda$ with $x \in U_i$. We consider the path $K(x) \in \mathcal{Q}_i$, that connects x to some vertex $x_1 \in U_{i_1}$, for some $0 \leq i_1 < i$. If $x' \in \beta$, then we terminate this process, and otherwise we consider the path $K(x_1)$, which is concatenated to path $K(x)$. After at most λ such iterations, we must obtain a path P, that connects x to some vertex $y \in \beta$. Since the length of every path in set $\bigcup_{j=1}^{\lambda-1} \mathcal{Q}_j$ is at most $4d' \log^{10} N$, the final length of the resulting path is bounded by $4\lambda d' \log^{10} N \leq 32d' \log^{11} N < d$, since $d' = \frac{d}{(\log N)^{64}}$, and N is sufficiently large. In time $O(d)$, we can turn this path into a simple path, and truncate it so that it still connects x to some vertex of β, but the inner vertices on the path do not belong to β. The time required to respond to a single query is bounded by $O(d)$, and the total time required to respond to all queries is bounded by $O(d \cdot \Delta) \leq O\left(n \cdot (n - |\beta^0| + \Delta) \cdot (\log N)^{64}\right)$, since, from the problem definition, $\Delta \cdot d \leq \eta \cdot \left(|V(G)| - |\beta| + \Delta \cdot (\log N)^{64}\right)$, and $\eta \leq \Delta \leq |\beta| \leq n$.

Recall that, in our setting, $\rho = O(\log^3 N)$, and so the running time of the algorithm for the ConnectToCenters problem from Theorem 6.7 is bounded by:

$$O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + |\beta'| + \Lambda) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+29}\right).$$

Since, from Claim 6.12, $\Lambda \leq 4n - 4|\beta^0| + 5\Delta \cdot (\log N)^{64}$ and $|\beta'| \leq 45n - 45|\beta^0| + 56\Delta \cdot (\log N)^{64}$, while $n = W^0$, we get that the total running time of the algorithm is bounded by:

$$\begin{aligned} O\left(c_1 \cdot W^0 \cdot (W^0 - |\beta^0| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+100}\right) \\ \leq c_1 \cdot W^0 \cdot (W^0 - |\beta| + \Delta) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r}. \end{aligned}$$

Finally, recall that the algorithm for the ConnectToCenters problem may terminate with a "FAIL", with probability at most $\frac{1}{N^7}$. If this happens, then our algorithm terminates with a "FAIL" as well, but the probability for this happening is bounded by $\frac{1}{N^7}$.

7 From the MaintainCluster Problem to RouteAndCut Problem – Comleting the Proof of Theorem 4.12

The goal of this section is to complete the proof of Theorem 4.12, by providing an algorithm for the MaintainCluster problem, that is summarized in the following theorem.

Theorem 7.1 *Suppose that, for some parameter $r \geq 1$, there exists a randomized algorithm for the r-restricted* RouteAndCut *problem, that, given as an input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most $c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2}$. Then there exists a randomized algorithm for the r-restricted* MaintainCluster *problem, that, on input (G, Δ, η, d, N) has running time at most: $c_1 \cdot (W^0(G))^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2 r} + c_1 \cdot W^0(G) \cdot \Delta \cdot \log^4 N$.*

We assume that there exists a randomized algorithm $\mathcal{A}$ for the r-restricted RouteAndCut problem, for some integer $r \geq 1$, that, on an input an instance $(G, A, B, \Delta, \eta, N)$ of the problem with $|V(G)| = n$, has running time at most $c_1 \cdot n \cdot (n - |B|) \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2}$.

Recall that the input to the r-restricted MaintainCluster problem consists of a perfect well-structured n-vertex graph $G = (L, R, E)$ with a proper assignment $\ell(e)$ of lengths to its edges. Recall that we have defined vertex weights for perfect well-structured graphs, where the weight $w(v)$ of a vertex v is the length of the unique special edge incident to v. For every vertex $v \in V(G)$, we denote by $w^0(v)$ its initial weight, and we denote $W^0(G) = \sum_{v \in V(G)} w^0(v)$. Additionally, we are given a parameter $N \geq W^0(G)$, that is greater than a large enough constant, with $r \leq \lceil \sqrt{\log N} \rceil$, and parameters $1 \leq \Delta \leq N$, $1 \leq \eta \leq \Delta$, and $(\log N)^{64} \leq d \leq 2^{r \cdot \sqrt{\log N}}$, such that $\frac{\Delta \cdot d}{\eta} \leq n$. In particular:

$$d \leq \frac{n\eta}{\Delta} \leq n. \tag{15}$$

Since $d \geq (\log N)^{64}$, we get that $(\log N)^{64} \leq n \leq N$, and so:

$$\log\log N \leq \log n \leq \log N. \tag{16}$$

Since we have assumed that N is greater than a large enough constant, and in particular that $N > 2^{2^{c_2}}$, we can also assume that n is greater than a large enough constant. We also get following inequality that will be useful for us later:

$$\log(16N) = 4 + \log N \leq \left(1 + \frac{1}{32c_2 r}\right) \log N. \tag{17}$$

Additionally, $\sqrt{\log(16N)} \leq \sqrt{4 + \log N} \leq 2 + \sqrt{\log N}$, and so:

$$2^{c_2\sqrt{\log(16N)}} \leq 2^{2c_2} \cdot 2^{c_2\sqrt{\log N}} \leq 2^{c_2\sqrt{\log N}} \cdot \log N, \tag{18}$$

since N is sufficiently large.

The algorithm consists of at most Δ iterations. At the beginning of every iteration i, the algorithm is given two vertices $x, y \in V(G)$, and it must return a simple path P_i in G connecting x to y, whose length is at most d. (We may sometimes say that the algorithm is given a short-path query between the pair (x, y) of vertices, and it returns the path P_i in response to the query). The lengths of some of the special edges on path P_i are then doubled. If e is a special edge of G, and $\tau < \tau'$ are consecutive times during the algorithm's execution when the length of e was doubled, then we are guaranteed that the number of the paths P_i that the algorithm returned in response to the queries during time interval $(\tau, \tau']$ with $e \in E(P_i)$ is at least η.

The algorithm may, at any time, produce a strongly well-structured cut (X, Y) in G of sparsity $\Phi_G(X, Y) \leq \frac{(\log N)^{64}}{d}$. If $w(X) \leq w(Y)$, then let $Z = X$ and $Z' = Y$; otherwise, let $Z = Y$ and $Z' = X$. We also let $J' \subseteq Z'$ denote the set of vertices that serve as endpoints of the edges of $E_G(X, Y)$. The vertices of $Z \cup J'$ are then deleted from G, and the algorithm continues with the resulting graph $G = G[Z' \setminus J']$. This modification ensures that G remains a perfect well-structured graph. Once $|V(G)| \leq n/2$ holds, the algorithm terminates, even if fewer than Δ iterations have passed. The algorithm may also, at any time, return "FAIL", but the probability of this ever happening must be bounded by $1/N^4$.

We can assume w.l.o.g. that, in the initial graph G, the length of every edge is at most d. Indeed, if (x, y) is an edge of G of length greater than d, then cut (A, B) with $A = \{x\}$ and $B = V(G) \setminus \{x\}$ is a strongly well-structured cut of sparsity at most $\frac{1}{d}$, since $w(x), w(y) = d$, and $|E_G(A, B)| = 1$. As long as special edges of length greater than d lie in G, we can therefore produce $\frac{1}{d}$-sparse cuts that will result in the removal of such edges from G. The time required to perform this preprocessing step is $O(|E(G)|) \leq O(W^0(G))$. We will also ensure that, throughout the algorithm, every edge of G has length at most $2d$. Therefore, if we denote by $W^{(\tau)}$ the total weight of all edges at some time τ during the algorithm's execution, then $W^{(\tau)} \leq 2n \cdot d \leq 2n^2$ must hold (we have used Inequality 15). Since the algorithm terminates once the number of vertices falls below $n/2$, the total weight W of all vertices of G satisfies $\frac{n}{2} \leq W \leq 2n^2$, at all times, and therefore, at all times the following inequality holds:

$$\log n - 1 \leq \log W \leq 4 \log n. \tag{19}$$

Throughout, we denote by G^0 the initial graph G, and by E^{spec} be the set of all special edges of G^0. For each such edge $e \in E^{\mathsf{spec}}$, let $\ell^0(e)$ be the length of e at the beginning of the algorithm, and let $\ell^*(e)$ be defined as follows. If e lies in graph G at the end of the algorithm, then $\ell^*(e)$ is the length of edge e at the end of the algorithm. Otherwise, $\ell^*(e)$ is the length of the edge e just before it was deleted from G. We will use the following simple observation to bound $\sum_{e \in E^{\mathsf{spec}}} \ell^*(e)$. The proof is deferred to Section D.1 of Appendix.

Observation 7.2

$$\sum_{e \in E^{\mathsf{spec}}} \ell^*(e) \leq 2 \sum_{e \in E^{\mathsf{spec}}} \ell^0(e) + \frac{2d\Delta}{\eta} \leq 2 \sum_{e \in E^{\mathsf{spec}}} \ell^0(e) + 2n.$$

Throughout the algorithm, instead of working with graph G, it will be convenient for us to work with its corresponding subdivided graph G^+ (see Definition 4.1). We denote the two sides of the bipartition of the vertices of G^+ by L^+ and R^+, respectively. Recall that graph G^+ is obtained from G by replacing every special edge $e = (u, v)$ with a new path, that we denote by $P^*(e)$, and call *the path represeinting e in G^+*. The number of special edges on path $P^*(e)$ is exactly $\ell(e)$. For every special edge e' on path $P^*(e)$, we say that e is the *parent-edge* of e', and that e' is the *child-edge* of e.

Assume that at some time during the algorithm's execution, the length $\ell(e)$ of an edge $e \in E(G)$ is doubled. Then for every special edge $e' = (x, y)$ on path $P^*(e)$, we replace the edge by a path (x, x', y', y), where x' and y' are new vertices, edges (x, x') and (y', y) are special, and edge (x', y') is regular; we call this process the *splitting* of edge e'. Notice that, since (x, y) is a special edge of G^+, $x \in R^+$ and $y \in L^+$ must hold. We add x' to L^+ and y' to R^+. It is easy to verify that G^+ remains the subdivided graph of G after this update.

In order to simplify the notation, we will denote G^+ by H from now on, and we will also denote the bipartition (L^+, R^+) by (L', R'). Recall that H is a uniform perfect well-structured graph. The initial number of vertices in H is W^0. As the algorithm progresses and edges of H are subdivided, we will denote by W the current number of vertices in H, which is equal to $\sum_{v\in V(G)} w(v)$.

From now on we will mostly work with graph H. In every iteration, we are given a pair (x, y) of vertices of H, and we need to return a simple path P connecting x to y in H, of length at most d. Such a path can be easily trasformed into a path in graph G of length at most d connecting x to y. After that, some special edges on path P are split. From the definition of the MaintainCluster problem, we are guaranteed that the following property holds throughout the algorithm.

Q'1. Consider any special edge e of H that did not lie in the original graph $H = (G^0)^+$. Assume that e was added to H at some time τ during the algorithm's execution, and was later split at some time $\tau' > \tau$ (so after time τ', edge e no longer lies in H). Then the number of paths P that the algorithm returned in response to queries during the time interval $(\tau, \tau']$ with $e \in E(P)$ is at least η.

Notice that $W^0 = 2\sum_{e\in E^{\mathsf{spec}}} \ell^0(e)$, and the total number of vertices that ever belonged to H over the course of the algorithm is bounded by $\sum_{e\in E^{\mathsf{spec}}} 2\ell^*(e) \leq 4\sum_{e\in E^{\mathsf{spec}}} \ell^0(e) + 4n \leq 2W^0 + 4n \leq 6W^0$, from Observation 7.2. In particular, $|V(H)| \leq 6W^0$ holds at all times.

Throughout, we use a parameter d', that is the smallest integral power of 2 that is greater than $\frac{d}{(\log N)^{28}}$, and we set $\varphi = \frac{4}{d'}$. Intuitively, if, at any time during the algorithm's execution, the length of any special edge $e = (x, y)$ in G becomes at least $d'/2$, then we can immediately produce a cut (A, B) with $A = \{x\}$ and $B = V(G) \setminus \{x\}$. It is easy to verify that (A, B) is a strongly well-structured cut in G. Moreover, since $w(x), w(y) = \ell(e)$, the sparsity of the cut is at most φ, and so our algorithm could output this cut. In reality our algorithm is somewhat more complex, but it uses a similar method to ensure that, at all times, the lengths of all edges in G are bounded by d'. Additionally, as long as the algorithm does not terminate, $\sum_{e\in E(G)} \ell(e) \geq \frac{|V(G)|}{2} \geq \frac{n}{4} > d'$ must hold.

Let W be the total weight of all vertices of G at any time during the algorithm's execution. Then $W \geq |V(G)| \geq \frac{n}{2} \geq \frac{d}{2}$ (from Inequality 15). Therefore, throughout the algorithm:

$$d' \leq \frac{2d}{\log^{28} N} \leq \frac{4W}{\log^{28} N} \tag{20}$$

holds. As long as the lengths of all edges in G are bounded by $d'/2$, our algorithm may produce, at any time, a weakly well-structured cut (A, B) in the current graph H, of sparsity at most $\frac{\varphi}{4}$. Let $A' = A \cap V(G)$ and $B' = B \cap V(G)$. From Observation 4.4, (A, B) is a weakly well-structured cut in G of sparsity at most 2φ. Furthermore, using the algorithm from Observation 4.5, we can transform cut (A, B) into a strongly well-structured cut (A^*, B^*) in G, of sparsity at most 4φ. We then return cut (A^*, B^*), following which graph G is updated with the deletion of the vertices on the side of the cut whose current weight is smaller, and the endpoints of the edges in $E_G(A^*, B^*)$. We update graph H by deleting from it every vertex that was deleted from G, and, additionally, for every edge e that was deleted from G, its corresponding path $P^*(e)$ is deleted from H. As before, the algorithm is allowed to halt and return "FAIL" at any time, but we are required that this happens with probability at most $1/N^4$.

7.1 Partition into Phases and Probability of Failure

Our algorithm is partitioned into phases. A phase terminates if one of the following happened: (i) the algorithm returned "FAIL", in which case we say that the phase is of *type 1*; or (ii) at least $\frac{\hat{W}}{\log^{30} N}$

vertices have been deleted from graph H during the current phase, where $\hat{W}$ is the number of vertices in H at the beginning of the phase, in which case we say that the phase is of *type 2*; or (iii) at least $\Delta' = \frac{\Delta}{\log^{30} N}$ queries have been responded to; in which case we say that the phase is of *type 3*. We will ensure that the probability that the algorithm returns "FAIL" in a single phase is bounded by $\frac{1}{2}$. Since the total number of vertices that ever belong to graph H is bounded by $6W^0 \leq O(N)$, we get that the total number of type-2 and type-3 phases is bounded by $O(\log^{30} N \cdot \log N) \leq \log^{32} N$. If $\lceil 16 \log N \rceil$ consecutive type-1 phases occur, then we terminate the algorithm and return "FAIL". Notice that, for each type-2 phase and type-3 phase, the probability that it is followed by $\lceil 16 \log N \rceil$ consecutive type-1 phases is bounded by $\left(\frac{1}{2}\right)^{\lceil 16 \log N \rceil} \leq \frac{1}{N^{16}}$. Therefore, overall, the probability that the algorithm ever returns "FAIL" is bounded by $\frac{(\log N)^{32}}{N^{16}} < \frac{1}{N^4}$, as required. The total number of phases in the algorithm is then bounded by $O(\log^{32} N \cdot \log N) \leq O(\log^{33} N)$.

At a very high level, our algorithm follows a somewhat standard framework, though the specific details are quite different. As a first step, we select a random subset T of vertices of H, that we call *terminals*, where every vertex of H is selected to be added to T independently with probability $\frac{1}{d'}$. Next, we will attempt to embed an expander X, whose vertex set is T, into graph H, so that every embedding path has length $\widetilde{O}(d')$, and the paths cause edge-congestion $O(\text{poly} \log n)$. We show that, with high probability, if we are unable to compute such an embedding, then we can compute a cut (A, B) in H, that can be transformed into a $\frac{(\log N)^{64}}{d}$-sparse strongly well-structured cut (A', B') in G with $w(A'), w(B') \geq \frac{\hat{W}}{\log^{30} N}$. We then output this cut, and update graphs G and H with the corresponding deletions of vertices and edges, which must lead to the deletion of at least $\frac{\hat{W}}{\log^{30} N}$ vertices from H. Therefore, with high probability, if we cannot successfully compute an expander X, and embed it into H, the current phase will terminate. Assume now that we have successfully computed an expander X over the set T of roughly $\frac{\hat{W}}{d'}$ vertices, and embedded it into H. Then we will employ the Layered Connectivity Data Structure from Section 6, with the set $V(X)$ of vertices serving as the set β of centers, in order to maintain a collection $\mathcal{Q}^{\text{out}}$ of short paths, connecting every vertex of $V(H)$ to some vertex of $V(X)$. We will also employ the same data structure on graph $\overleftarrow{G}$, that is obtained from G by reversing the direction of each of its edges, to maintain a collection $\mathcal{Q}^{\text{in}}$ of short paths, that, for every vertex $v \in V(H)$, contains a path that connects some vertex of $V(X)$ to v.

We now formally describe the algorithm for executing a single phase. We assume w.l.o.g. that, at the beginning of the phase, graph G does not contain any edge of length greater than d'. Otherwise, as long as such an edge $e = (x, y)$ lies in G, then cut (A, B) with $A = \{x\}$ and $B = V(G) \setminus \{x\}$ is a strongly well-structured cut of sparsity $\frac{1}{d'} \leq \varphi$, since $w(x) = d'$, and $|E_G(A, B)| = 1$. As long as special edges of length greater than d' lie in G, we can therefore produce φ-sparse cuts that will result in the removal of such edges from G. The time required to perform this preprocessing step, including updating the graph H, is $O(|V(H)|) \leq O(W^0)$. Therefore, we assume that, at the beginning of the phase, G does not contain edges of length greater than d'. We denote by $\hat{W}$ the number of vertices in H at the beginning of the phase.

We start with a preprocessing step, in which we sample the vertices of H to be added to the set T. Then we describe our two main data structures. The first data structure maintains a large expander that is defined over the set T of vertices, while the second data structure is Layered Connectivity Data Structure in graphs G and $\overleftarrow{G}$.

7.2 Preprocessing Step: Subsampling

Recall that the number of vertices in H at the beginning of the phase, denoted by $\hat{W}$, is bounded by $12W^0 \leq 12N$.

In this step, we construct a set T of vertices of H, that we call *terminals*, as follows: every vertex $v \in V(H)$ is added to T with probability $\frac{1}{d'}$ independently. This is the only randomized part in our algorithm for the Maximum Bipartite Matching problem. Next, we will define four "bad" events, and we will show that each of them may only happen with low probability. Eventually, if our algorithm returns "FAIL" and halts, we will guarantee that at least one of these bad events must have happened. This will ensure that the probability that the phase terminates with the "FAIL" is at most $\frac{1}{2}$, as required.

7.2.1 Bad Event $\mathcal{E}_1$

Observe first that the expected cardinality of T is $\frac{\hat{W}}{d'}$. We say that bad event $\mathcal{E}_1$ happens if $|T| > \frac{8\hat{W}}{d'}$ or $|T| < \frac{\hat{W}}{2d'}$. From Lemma 2.5, $\mathbf{Pr}\left[|T| > \frac{8\hat{W}}{d'}\right] \leq 2^{-8\hat{W}/d'} \leq \frac{1}{512}$, and $\mathbf{Pr}\left[|T| < \frac{\hat{W}}{2d'}\right] \leq e^{-\hat{W}/(8d')} \leq \frac{1}{512}$, since $d' \leq \frac{4\hat{W}}{\log^{28} N}$ from Inequality 20. Overall, we get that $\mathbf{Pr}\left[\mathcal{E}_1\right] \leq \frac{1}{256}$, and, if $\mathcal{E}_1$ does not happen, then $\frac{\hat{W}}{2d'} \leq |T| \leq \frac{8\hat{W}}{d'}$ holds.

Before we define the remaining two bad events, we need to inspect the well-structured cuts of H more closely.

7.2.2 Well-Structured Cuts and Their Structure

Recall that a cut (A, B) of H is weakly well-structured iff all edges of $E_H(A, B)$ are special. Let E' be any subset of special edges of H. We denote by $\mathcal{C}(E')$ the collection of all weakly well-structured cuts (A, B) in H with $E_H(A, B) = E'$. Next, we investigate the structure of the cuts in $\mathcal{C}(E')$.

Let $J(E')$ denote the set of all vertices $v \in V(H)$, such that some edge of E' leaves v, and let $J'(E')$ denote the set of all vertices $u \in V(H)$, such that some edge of E' enters u. Next, we denote by $S^*(E')$ the set of all vertices $v \in H$, such that there is a path in $H \setminus E'$ connecting a vertex of $J(E')$ to v. Notice that, for every cut $(A, B) \in \mathcal{C}(E')$, it must be the case that $S^*(E') \subseteq A$. Indeed, assume otherwise, and let $v \in S^*(E')$ be any vertex that lies in B. Since $J(E') \subseteq A$ must hold, there is a path P connecting a vertex of A to v, such that P is disjoint from the edges of E'. But then some edge of P must lie in $E_H(A, B)$, contradicting the fact that $E_H(A, B) = E'$.

Similarly, we denote by $S^{**}(E')$ the set of all vertices $v \in H$, such that there is a path in $H \setminus E'$ connecting v to a vertex of $J'(E')$. From the same arguments as above, for every cut $(A, B) \in \mathcal{C}(E')$, $S^{**}(E') \subseteq B$ must hold.

Let $\mathcal{X}'$ be the set of all strongly connected components of $H \setminus (S^*(E') \cup J(E') \cup S^{**}(E') \cup J'(E'))$, where for each $X \in \mathcal{X}'$, we view X as a set of vertices corresponding to the connected component. Denote $\mathcal{X}(E') = \mathcal{X}' \cup \{S^*(E') \cup J(E'), S^{**}(E') \cup J'(E')\}$. Clearly, $\mathcal{X}(E')$ is a partition of $V(H)$. Consider again any cut $(A, B) \in \mathcal{C}(E')$. As observed already, $S^*(E') \cup J(E') \subseteq A$ and $S^{**}(E') \cup J'(E') \subseteq B$ must hold. Moreover, it is easy to see that, since $E_H(A, B) = E'$, for all $X \in \mathcal{X}'$, either $X \subseteq A$, or $X \subseteq B$ must hold. From the definition of the sets $S^*(E'), S^{**}(E')$ of vertices, graph $H \setminus E'$ contains no path connecting a vertex of $S^*(E') \cup J(E')$ to a vertex lying outside this set. Similarly, no path in $H \setminus E'$ may connect a vertex of $V(H) \setminus (S^{**}(E') \cup J'(E'))$ to a vertex of $S^{**}(E') \cup J'(E')$. It is now easy to verify that, for every pair $X, X' \in \mathcal{X}(E')$ of sets, it cannot be the case that $H \setminus E'$ contains both a path P connecting a vertex of X to a vertex of X', and a path P' connecting a vertex of X' to a vertex of X. Indeed, if such paths did exist, then neither of these paths could contain vertices of $S^*(E') \cup J(E') \cup S^{**}(E') \cup J'(E')$, and so the vertices of $X \cup X'$ should have belonged to a single strongly connected component of $H \setminus (S^*(E') \cup J(E') \cup S^{**}(E') \cup J'(E'))$.

We can now define an arbitrary topological ordering $(X_1, X_2, \ldots, X_q)$ of the sets in $\mathcal{X}(E')$, such that

$X_1 = S^{**}(E') \cup J(E')$, $X_q = S^*(E') \cup J'(E')$, and for all $1 \le i < j \le q$, there is no edge of $E(H) \setminus E'$ that connects a vertex of X_j to a vertex of X_i. Notice that, for all $1 \le i \le q$, we can define a cut (A_i, B_i) with $A_i = X_{i+1} \cup \cdots \cup X_q$ and $B_i = X_1 \cup \cdots \cup X_i$. From our discussion so far, the only edges that connect vertices of A_i to vertices of B_i are the edges of E', and so for each such $1 \le i < r$, cut (A_i, B_i) lies in $\mathcal{C}(E')$. We call all cuts in $\{(A_i, B_i) \mid 1 \le i \le q\}$ *canonical cuts* corresponding to E', and denote the set of all such cuts by $\mathcal{C}'(E')$. Notice that $\mathcal{C}'(E') \subseteq \mathcal{C}(E')$, and it is possible that $\mathcal{C}(E') \setminus \mathcal{C}'(E') \neq \emptyset$. Note that the definition of the set $\mathcal{C}'(E')$ of cuts depends on the topological ordering of the sets in $\mathcal{X}(E')$, and there may be several such topological orderings. We assume that we are given some tie-breaking rule that selects a single topological ordering of the sets in $\mathcal{X}(E')$, and we define $\mathcal{C}'(E')$ with respect to that topological ordering. We also assume that the above unique topological ordering can be computed in time $O(n)$, given the sets in $\mathcal{X}(E')$. We are now ready to define the second bad event.

7.2.3 Bad Event $\mathcal{E}_2$

We start with intuition. Recall that our goal is to embed an expander over the set T of vertices into H, with polylogarithmic congestion. Intuitively, if we are not able to do so, we will construct a cut (A, B) in H, such that $|T \cap A|, |T \cap B| \ge \Omega(|T|/\operatorname{poly}\log N)$, and $|E_H(A, B)|$ is sufficiently low. If this happens, we would like to be able to compute a sparse and a roughly balanced cut in graph H. If we denote by $E' = E_H(A, B)$, then ideally we would like to ensure that some canonical cut corresponding to E' has this property. The second bad event happens if this is not the case for some such cut (A, B). Next, we formally define the second bad event, and show that it can only happen with small probability.

We say that bad event $\mathcal{E}_2$ happens if there is a weakly well-structured cut (A, B) in H, with $|A \cap T|, |B \cap T| \ge \frac{|T|}{\log^4 N}$, and $|E_H(A, B)| \le \frac{|T|}{\log^6 N}$, such that, if we denote by $E' = E_H(A, B)$, then for every canonical cut $(A', B') \in \mathcal{C}'(E')$, either $|A'| \le \frac{\hat{W}}{\log^5 N}$, or $|B'| \le \frac{\hat{W}}{\log^5 N}$ holds.

Claim 7.3

$$\mathbf{Pr}\left[\mathcal{E}_2\right] \le \frac{1}{64}.$$

Proof: Recall that, if event $\mathcal{E}_1$ does not happen, then $\frac{\hat{W}}{2d'} \le |T| \le \frac{8\hat{W}}{d'}$ holds, and so, if E' is a set of special edges with $|E'| \le \frac{|T|}{\log^6 N}$, then $|E'| \le \frac{8\hat{W}}{d' \log^6 N}$.

For every subset E' of at most $\frac{8\hat{W}}{d' \log^6 N}$ special edges, we define a bad event $\mathcal{E}(E')$ to be the event that there is some cut (A, B) in H with $E_H(A, B) = E'$, such that $|A \cap T|, |B \cap T| \ge \frac{|T|}{\log^4 N}$, and for every canonical cut $(A', B') \in \mathcal{C}'(E')$, either $|A'| \le \frac{\hat{W}}{\log^5 N}$, or $|B'| \le \frac{\hat{W}}{\log^5 N}$ holds. We bound $\mathbf{Pr}\left[\mathcal{E}(E')\right]$ for each such set E' of edges separately, and then use the union bound in order to bound $\mathbf{Pr}\left[\mathcal{E}_2\right]$.

Fix any subset E' of at most $\frac{8\hat{W}}{d' \log^6 N}$ special edges. Let $X \in \mathcal{X}(E')$ be the set of vertices with maximum cardinality. We claim that Event $\mathcal{E}(E')$ may only happen if $|X| \ge \hat{W} - \frac{2\hat{W}}{\log^5 N}$, in the following simple observation.

Observation 7.4 *If $\mathcal{E}(E')$ happened, then $|X| \ge \hat{W} - \frac{2\hat{W}}{\log^5 N}$ must hold.*

Proof: Assume otherwise, that is, Event $\mathcal{E}(E')$ happened, but $|X| < \hat{W} - \frac{2\hat{W}}{\log^5 N}$. Denote $\mathcal{X}(E') = (X_1, X_2, \ldots, X_q)$, where the sets are listed according to their topological order, and assume that $X = X_i$.

Assume first that $|X_i| \geq \frac{\hat{W}}{4}$. Then either $|\bigcup_{i'<i} X_{i'}| \geq \frac{\hat{W}}{\log^5 N}$, or $|\bigcup_{i'>i} X_{i'}| \geq \frac{\hat{W}}{\log^5 N}$. In the former case, cut (A_i, B_i) is a canonical cut with $|A_i|, |B_i| \geq \frac{\hat{W}}{\log^5 N}$, while in the latter case, cut (A_{i-1}, B_{i-1}) is a canonical cut with $|A_{i-1}|, |B_{i-1}| \geq \frac{\hat{W}}{\log^5 N}$, a contradiction (if $i = 1$ then we are guaranteed that (A_i, B_i) is the cut with the desired properties, and if $i = q$, then we are guaranteed that (A_{q-1}, B_{q-1}) is such a cut).

Assume next that $|X_i| < \frac{\hat{W}}{4}$. In this case, we let $1 < j < q$ be the smallest index for which $\sum_{j'<j} |X_{j'}| \geq \frac{\hat{W}}{2}$. It is easy to verify that $\sum_{j'<j} |X_{j'}| \leq \frac{3\hat{W}}{4}$ must hold, and so cut $(A_j, B_j) \in \mathcal{C}'(E')$ satisfies that both $|A_j|, |B_j| \geq \frac{\hat{W}}{\log^5 N}$, a contradiction. □

We assume from now on that $|X| \geq \hat{W} - \frac{2\hat{W}}{\log^5 N}$ holds for the set E' of edges. Let $Y = V(H) \setminus X$, so $|Y| \leq \frac{2\hat{W}}{\log^5 N}$. Consider now any cut $(A, B) \in \mathcal{C}(E')$, and assume that $|A \cap T|, |B \cap T| \geq \frac{|T|}{\log^4 N}$. Notice that all vertices of X must lie on one side of the cut (A, B). If $X \subseteq A$, then $B \subseteq Y$, and, since $|T \cap B| \geq \frac{|T|}{\log^4 N}$, we get that $|T \cap Y| \geq \frac{|T|}{\log^4 N}$. Similarly, if $X \subseteq B$, then $A \subseteq Y$, and, since $|T \cap A| \geq \frac{|T|}{\log^4 N}$, we get that $|T \cap Y| \geq \frac{|T|}{\log^4 N}$. In either case, we conclude that Event $\mathcal{E}(E')$ may only happen if $|X| \geq \hat{W} - \frac{2\hat{W}}{\log^5 N}$ and $|T \cap Y| \geq \frac{|T|}{\log^4 N}$.

We start by bounding the probability that $|T \cap X| < \frac{\hat{W}}{4d'}$. Since we have assumed that $|X| \geq \hat{W} - \frac{2\hat{W}}{\log^5 N} \geq \frac{\hat{W}}{2}$, and since every vertex is selected into T independently with probability $1/d'$, we get that $\mathbf{E}\left[|T \cap X|\right] \geq \frac{\hat{W}}{2d'}$. From Lemma 2.5, $\mathbf{Pr}\left[|T \cap X| < \frac{\hat{W}}{4d'}\right] \leq e^{-\hat{W}/(16d')}$.

Assume now that $|T \cap X| \geq \frac{\hat{W}}{4d'}$, so in particular $|T| \geq \frac{\hat{W}}{4d'}$. In this case, if $|T \cap Y| \geq \frac{|T|}{\log^4 N}$, then $|T \cap Y| \geq \frac{\hat{W}}{4d' \log^4 N}$ must hold. Since $\mathbf{E}\left[|T \cap Y|\right] = \frac{|Y|}{d'} \leq \frac{2\hat{W}}{d' \log^5 N}$, from Lemma 2.5, $\mathbf{Pr}\left[|T \cap Y| \geq \frac{\hat{W}}{4d' \log^4 N}\right] \leq 2^{-\hat{W}/(4d' \log^4 N)} \leq e^{-\hat{W}/(16d')}$.

Altogether, we get that:

$$\mathbf{Pr}\left[\mathcal{E}(E')\right] \leq \mathbf{Pr}\left[|T \cap X| < \frac{\hat{W}}{4d'}\right] + \mathbf{Pr}\left[|T \cap Y| \geq \frac{|T|}{\log^4 N} \mid |T \cap X| \geq \frac{\hat{W}}{4d'}\right] \leq 2 \cdot e^{-\hat{W}/(16d')}.$$

Observe that H has $\hat{W}$ special edges, and so the number of sets E' containing at most $\frac{8\hat{W}}{d' \log^6 N}$ special edges is bounded by:

$$(2\hat{W})^{8\hat{W}/(d' \log^6 N)} \leq (24N)^{8\hat{W}/(d' \log^6 N)} \leq 2^{32\hat{W}/(d' \log^5 N)}.$$

(We have used the fact that $\hat{W} \leq 12\hat{W}^0 \leq 12N$). Using the union bound over all sets E' of at most $\frac{8\hat{W}}{d' \log^6 N}$ special edges, we get that the probability that event $\mathcal{E}(E')$ happens for any such set of edges is bounded by:

$$2^{\frac{32\hat{W}}{d' \log^5 N}} \cdot 2 \cdot 2^{-\frac{\hat{W}}{16d' \log^4 N}} \leq 2^{-\frac{\hat{W}}{32d' \log^4 N}} \leq \frac{1}{512},$$

since $d' \leq \frac{8\hat{W}}{\log^{28} N}$ from Inequality 20.

Finally, if Event $\mathcal{E}_2$ happened, then either $\mathcal{E}_1$ happened, or, for some set E' of at most $\frac{8\hat{W}}{d' \log^6 N}$ special edges, event $\mathcal{E}(E')$ has happened. Therefore, overall, $\mathbf{Pr}\left[\mathcal{E}_2\right] \leq \frac{1}{256} + \frac{1}{512} \leq \frac{1}{64}$. □

7.2.4 Bad Event $\mathcal{E}_3$

We start again with intuition. Once we construct the expander X and embed it into H, we will need to route the remaining vertices of H to the vertices of T via paths that cause congestion roughly $\widetilde{O}(d')$. If we are unable to do so, we will compute a sparse cut that separates some vertices of H from most terminals in T. Denote this cut by (A, B). The bad scenairo that we would like to avoid is when one side of the cut (say A) contains almost all vertices of H, while the other side of the cut contains few vertices of H, but it contains most terminals. If this happens, then the vertices of B will be subsequently deleted from H, thereby destroying the expander that we have constructed. As we need to invest a large amount of time into constructing the expander X and embedding it into H, we would like to avoid such a situation. Let $E' = E_G(A, B)$. If some canonical cut $(A', B') \in \mathcal{C}'(E')$ is roughly balanced with respect to the vertices of H, then we can return cut (A', B') instead of cut (A, B), and ensure that a sufficiently large number of vertices is deleted from H, and the current phase can be terminated. However, if $\mathcal{C}'(E')$ contains no such cut, then it is not clear how to bypass the problem mentioned above. The third bad event is designed to capture exactly this scenario. We now formally define it and show that it is unlikely to happen.

We say that the bad event $\mathcal{E}_3$ happens if there is a weakly well-structured cut (A, B) in H, with $|E_H(A, B)| \leq \frac{|T|}{\log^6 N}$, such that one of the sides of the cut contains at least $\frac{\hat{W}}{2}$ vertices of H and at most $\frac{|T|}{32}$ vertices of T, and additionally, if we denote by $E' = E_H(A, B)$, then for every canonical cut $(A', B') \in \mathcal{C}'(E')$, either $|A'| \leq \frac{\hat{W}}{\log^5 N}$, or $|B'| \leq \frac{\hat{W}}{\log^5 N}$ holds.

Claim 7.5

$$\mathbf{Pr}\left[\mathcal{E}_3\right] \leq \frac{1}{64}.$$

Proof: As before, if event $\mathcal{E}_1$ does not happen, and E' is a set of special edges with $|E'| \leq \frac{|T|}{\log^6 N}$, then $|E'| \leq \frac{8\hat{W}}{d' \log^6 N}$ must hold. For every subset E' of at most $\frac{8\hat{W}}{d' \log^6 N}$ special edges, we define a bad event $\mathcal{E}'(E')$ to be the event that there is some weakly well-structured cut (A, B) in H with $E_H(A, B) = E'$, such that one side of the cut cut contains at least $\hat{W}/2$ vertices of H and at most $\frac{|T|}{32}$ vertices of T, and additionally, for every canonical cut $(A', B') \in \mathcal{C}'(E')$, either $|A'| \leq \frac{\hat{W}}{\log^5 N}$, or $|B'| \leq \frac{\hat{W}}{\log^5 N}$ holds. We bound $\mathbf{Pr}\left[\mathcal{E}'(E')\right]$ for each such set E' of edges separately, and then use the union bound in order to bound $\mathbf{Pr}\left[\mathcal{E}_3\right]$.

Fix any subset E' of at most $\frac{8\hat{W}}{d' \log^6 N}$ special edges. Let $X \in \mathcal{X}(E')$ be the set of vertices with maximum cardinality. Using the same arguments as in the proof of Claim 7.3, Event $\mathcal{E}'(E')$ may only happen if $|X| \geq \hat{W} - \frac{2\hat{W}}{\log^5 N}$. We assume from now on that $|X| \geq \hat{W} - \frac{2\hat{W}}{\log^5 N}$ holds. Let $Y = V(H) \setminus X$, so that $|Y| \leq \frac{2\hat{W}}{\log^5 N}$.

Consider now some weakly well-structured cut (A, B) in H with $E_H(A, B) = E'$, and assume that one of the sets $Z \in \{A, B\}$ contains at least $\hat{W}/2$ vertices of H and at most $\frac{|T|}{32}$ vertices of T. Since set X must be contained in one of the sides of the cut, it must be the case that $X \subseteq Z$, and $|T \cap X| \leq \frac{|T|}{32} = \frac{|T \cap X| + |T \cap Y|}{32}$. Therefore, $|T \cap X| \leq \frac{|T \cap Y|}{31}$ must hold.

Using the same reasoning as in the proof of Claim 7.3, $\mathbf{Pr}\left[|T \cap X| < \frac{\hat{W}}{4d'}\right] \leq e^{-\hat{W}/(16d')}$.

Assume now that $|T\cap X| \geq \frac{\hat{W}}{4d'}$. In this case, if $|T\cap X| \leq \frac{|T|}{32}$ holds, then $|T\cap Y| \geq 31|T\cap X| \geq \frac{31\hat{W}}{4d'}$ must hold. Notice that, since $|Y| \leq \frac{2\hat{W}}{\log^5 N}$, $\mathbf{E}\left[|T\cap Y|\right] \leq \frac{2\hat{W}}{d'\log^5 N}$. From Lemma 2.5, $\mathbf{Pr}\left[|T\cap Y| \geq \frac{31\hat{W}}{4d'}\right] \leq 2^{-31\hat{W}/(4d')} \leq e^{-\hat{W}/(16d')}$.

Altogether, we get that:

$$\mathbf{Pr}\left[\mathcal{E}'(E')\right] \leq \mathbf{Pr}\left[|T\cap X| < \frac{\hat{W}}{4d'}\right] + \mathbf{Pr}\left[|T\cap X| \leq \frac{|T|}{32} \mid |T\cap X| \geq \frac{\hat{W}}{4d'}\right] \leq 2\cdot e^{-\hat{W}/(16d')}.$$

As in the proof of Claim 7.3, the number of sets E' containing at most $\frac{8\hat{W}}{d'\log^6 N}$ special edges is bounded by $2^{32\hat{W}/(d'\log^5 N)}$.

Using the union bound over all sets E' of at most $\frac{8\hat{W}}{d'\log^6 N}$ special edges, we get that the probability that event $\mathcal{E}'(E')$ happens for any such set of edges is bounded by:

$$2^{\frac{32\hat{W}}{d'\log^5 N}}\cdot 2\cdot 2^{-\frac{\hat{W}}{16d'\log^4 N}} \leq 2^{-\frac{\hat{W}}{32d'\log^4 N}} \leq \frac{1}{512},$$

since $d' \leq \frac{8\hat{W}}{\log^{28} N}$ from Inequality 20.

Finally, if Event $\mathcal{E}_3$ happened, then either event $\mathcal{E}_1$ happened, or, for some set E' of at most $\frac{8\hat{W}}{d'\log^6 N}$ special edges, event $\mathcal{E}'(E')$ has happened. Therefore, overall, $\mathbf{Pr}\left[\mathcal{E}_3\right] \leq \frac{1}{256} + \frac{1}{512} \leq \frac{1}{64}$. □

7.2.5 Bad Event $\mathcal{E}_4$

Recall that for every edge $e \in E(G)$, there is a corresponding path $P^*(e)$, containing $2\ell(e)$ vertices, that represents e in graph H. Recall that we have ensured that, at the beginning of the phase, the length of every edge $e \in E(G)$ is at most d'. The fourth bad event, $\mathcal{E}_4$, happens if there is some edge $e \in E(G)$, such that at least $16\log N$ vertices of $P^*(e)$ are in T.

Claim 7.6 $\mathbf{Pr}\left[\mathcal{E}_4\right] \leq \frac{1}{64}$.

Proof: For every special edge $e \in E(G)$, let $\mathcal{E}''(e)$ be the bad event that at least $16\log N$ vertices of $P^*(e)$ are added to T. Since the total number of vertices on $P^*(e)$ is $2\ell(e) \leq 2d'$, and every edge is chosen to be added to T with probability $\frac{1}{d'}$, the expectation of $|P^*(e)\cap T|$ is at most 2. From Lemma 2.5, the probability of $\mathcal{E}''(e)$ is then bounded by $2^{-16\log N} \leq N^{-16}$.

By using the union bound over all special edges $e \in E(G)$, and since the number of such edges is bounded by $n \leq N$, we get that:

$$\mathbf{Pr}\left[\mathcal{E}_4\right] \leq N\cdot N^{-16} \leq \frac{1}{64}.$$

□

Lastly, we let $\mathcal{E}$ be the bad event that either of the events $\mathcal{E}_1, \mathcal{E}_2$, $\mathcal{E}_3$, or $\mathcal{E}_4$ happened. Clearly:

$$\mathbf{Pr}\left[\mathcal{E}\right] \leq \sum_{i=1}^{4}\mathbf{Pr}\left[\mathcal{E}_i\right] \leq \frac{1}{16}.$$

Note that we can check, in time $O(N)$, whether events $\mathcal{E}_1$ and $\mathcal{E}_4$ hold. If either of these events holds, we terminate the algorithm and return "FAIL". Therefore, we assume from now on that neither of the events $\mathcal{E}_1, \mathcal{E}_4$ happened.

As the phase progresses, vertices may be deleted from graphs G and H. We view the set T of terminal vertices as static, that is, it does not change even as some vertices of T are deleted from H.

7.3 Finalizing Cuts

In this subsection we define a special kinds of cuts in graph H, that we call *finalizing* cuts. If, at any time during the execution of a phase, our algorithm computes such a cut in the current graph H, we will be able to terminate the current phase, by either correctly establishing that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ must have happened, or by providing a strongly well-structured φ-sparse cut (A, B) in the current graph G, such that $w(A), w(B) \geq \frac{\hat{W}}{\log^{30} N}$. In the former case, we will terminate the current phase with a "FAIL", while in the latter case, once cut (A, B) is processed, and the corresponding vertices and edges are deleted from H, we are guaranteed that at least $\frac{\hat{W}}{\log^{30} N}$ vertices have been deleted from H since the beginning of the phase, and so the current phase terminates. We are now ready to define finalizing cuts in H.

Definition 7.7 (Finalizing Cut) *Let (A, B) be a cut in graph H at any time during the execution of a phase. We say that the cut (A, B) is a* finalizing *cut, if the following hold:*

- *all edges in $E_H(A, B)$ are special edges;*
- *$|E_H(A, B)| \leq \frac{|T|}{8 \log^6 N}$; and*
- *one of the sides of the cut $Z \in \{A, B\}$ contains at least $|V(H)| - \frac{\hat{W}}{32}$ vertices of H, and at most $|T| \cdot \left(1 - \frac{7}{\log^4 N}\right)$ vertices of T.*

Next, we show that, if our algorithm ever produces a finalizing cut during a phase, then the phase can be terminated, in the following claim.

Claim 7.8 *Suppose, at any time during the execution of a phase, there is a finalizing cut $(\tilde{A}, \tilde{B})$ in the current graph H. Assume further that neither of the events $\mathcal{E}_2, \mathcal{E}_3$ has happened, and that every edge in the current graph G has length at most $d'/2$. Then there is a strongly well-structured φ-sparse cut (A^*, B^*) in the current graph G, such that $w(A^*), w(B^*) \geq \frac{\hat{W}}{\log^{30} N}$. Moreover, there is a deterministic algorithm, that, given the graph G at the beginning of the current phase (that we denote by G_0), the sequence of the strongly well-structured φ-sparse cuts $(A_1, B_1), \ldots (A_q, B_q)$ in G that the algorithm has produced during the current phase so far, the current graphs G and H, and the finalizing cut $(\tilde{A}, \tilde{B})$ in the current graph H, either correctly estabishes that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ has happened, or computes a strongly well-structured φ-sparse cut (A^*, B^*) in the current graph G, such that $w(A^*), w(B^*) \geq \frac{\hat{W}}{\log^{30} N}$. The running time of the algorithm is $O(\hat{W}^2)$.*

The remainder of this subsection is dedicated to the proof of Claim 7.8. We denote by G_0 the graph G at the beginning of the current phase, and by H_0 its corresponding subdivided graph H. The main idea of the proof is to exploit the cuts $(A_1, B_1), \ldots (A_q, B_q)$ in graph G, together with the finalizing cut $(\tilde{A}, \tilde{B})$ in graph H, in order to compute another well-structured cut $(\tilde{S}, \tilde{S}')$ in H_0, such that $|E_H(\tilde{S}, \tilde{S}')| \leq \frac{|T|}{\log^6 N}$, and one side of the cut contains all but at most $\frac{\hat{W}}{16}$ vertices of H_0, but only a relatively small fraction of the vertices of T. Let $E'' = E_H(\tilde{S}, \tilde{S}')$. By inspecting all canonical cuts in $\mathcal{C}'(E'')$, we will then either correctly establish that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ happened, or we will compute a cut (A^*, B^*) in G with the required properties.

In the following technical claim, whose proof is deferred to Section D.2 of Appendix, we provide an algorithm for computing the cut $(\tilde{S}, \tilde{S}')$ in H_0 with the desired properties.

Claim 7.9 *There is a deterministic algorithm, that, given the graph G_0, the sequence of the strongly well-structured φ-sparse cuts $(A_1, B_1), \dots (A_q, B_q)$ in G that the algorithm has produced during the current phase so far, the current graphs G and H, such that all edge lengths in G are bounded by $d'/2$, and the finalizing cut $(\tilde{A}, \tilde{B})$ in the current graph H, computes a weakly well-structured cut $(\tilde{S}, \tilde{S}')$ in H_0, such that $|E_{H_0}(\tilde{S}, \tilde{S}')| \le \frac{|T|}{\log^6 N}$, and one side $\tilde{Z} \in \left\{\tilde{S}, \tilde{S}'\right\}$ of the cut contains all but at most $\hat{W}/8$ vertices of H_0, and at most $\left(1 - \frac{2}{\log^4 N}\right) \cdot |T|$ vertices of T. The running time of the algorithm is $O(\hat{W}^2)$.*

We are now ready to complete the proof of Claim 7.8. Let $E'' = E_{H_0}(\tilde{S}, \tilde{S}')$. We construct, in time $O(|E(H_0)|) \le O(\hat{W}^2)$, the collection $\mathcal{X}(E'')$ of subsets of H_0 associated with the set E'' of edges, and then examine all canonical cuts $(A, B) \in \mathcal{C}'(E'')$. For each such cut, we check whether $|A| \le \frac{\hat{W}}{\log^5 N}$, or $|B| \le \frac{\hat{W}}{\log^5 N}$ hold. From the definition of canonical cuts, we can check this, for every cut $(A, B) \in \mathcal{C}'(E'')$, in time $O(|V(H)|) \le O(\hat{W})$. We first claim that if, for each such canonical cut (A, B), either $|A| \le \frac{\hat{W}}{\log^5 N}$ or $|B| \le \frac{\hat{W}}{\log^5 N}$ holds, then at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ must have happened.

Observation 7.10 *If, for every canonical cut $(A, B) \in \mathcal{C}'(E'')$, either $|A| \le \frac{\hat{W}}{\log^5 N}$ or $|B| \le \frac{\hat{W}}{\log^5 N}$ holds, then at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ must have happened.*

Proof: Recall that one side $\tilde{Z} \in \left\{\tilde{S}, \tilde{S}'\right\}$ of the cut $(\tilde{S}, \tilde{S}')$ in H contains all but at most $\hat{W}/8$ vertices of H_0, and at most $\left(1 - \frac{2}{\log^4 N}\right) \cdot |T|$ vertices of T. Denote the other side of the cut by $\tilde{Z}'$. We now consider two cases. The first case happens if $|\tilde{Z} \cap T| \le \frac{|T|}{32}$. Then $\tilde{Z}$ contains at least $\frac{\hat{W}}{2}$ vertices of H and at most $\frac{|T|}{32}$ vertices of T. Since, for every canonical cut $(A, B) \in \mathcal{C}'(E'')$, either $|A| \le \frac{\hat{W}}{\log^5 N}$ or $|B| \le \frac{\hat{W}}{\log^5 N}$ holds, we conclude that Event $\mathcal{E}_3$ must have happened.

Otherwise, $|\tilde{Z} \cap T| \ge \frac{|T|}{32}$, and, since $|\tilde{Z} \cap T| \le \left(1 - \frac{2}{\log^4 N}\right) \cdot |T|$, we get that $|\tilde{Z} \cap T|, |\tilde{Z}' \cap T| \ge \frac{|T|}{\log^4 N}$. Since, for every canonical cut $(A, B) \in \mathcal{C}'(E'')$, either $|A| \le \frac{\hat{W}}{\log^5 N}$ or $|B| \le \frac{\hat{W}}{\log^5 N}$ holds, we conclude that Event $\mathcal{E}_2$ must have happened. □

If, for every canonical cut $(A, B) \in \mathcal{C}'(E'')$, either $|A| \le \frac{\hat{W}}{\log^5 N}$ or $|B| \le \frac{\hat{W}}{\log^5 N}$ hold, then we terminate the algorithm, and report that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ has happened. We assume from now on that there is a canonical cut $(A, B) \in \mathcal{C}'(E'')$ with $|A|, |B| \ge \frac{\hat{W}}{\log^5 N}$. Recall that (A, B) is a weakly canonical cut in graph H_0, and $|E_{H_0}| \le \frac{|T|}{\log^6 N} \le \frac{8\hat{W}}{d' \log^6 N} \le \frac{8}{d' \log N} \cdot \min\{|A|, |B|\} \le \frac{\varphi}{64} \min\{|A|, |B|\}$. Therefore, (A, B) is a weakly canonical cut of sparsity at most $\frac{\varphi}{64}$ in H_0. Recall also that every edge in G_0 has length at most $\frac{d'}{2} = \frac{2}{\varphi}$.

For every vertex $v \in V(G_0)$, we denote the weight of v in G_0 by $w_0(v)$, and, for every subset $S \subseteq V(G_0)$ of vertices, we denote $w_0(S) = \sum_{v \in S} w_0(v)$. Similarly, for every vertex $v \in V(G)$, we denote the weight of v in G by $w(v)$, and for a subset $S \subseteq V(G)$ of vertices, we denote $w(S) = \sum_{v \in S} w(v)$. Note that, for every vertex $v \in V(G)$, $w_0(v) \le w(v)$.

Consider now the cut (A', B') in G_0, where $A' = A \cap V(G_0)$ and $B' = B \cap V(G_0)$. From Observation 4.4, (A', B') is a weakly well-structured cut in G_0, whose sparsity is at most $\frac{\varphi}{8}$, with $w_0(A') \geq \frac{|A|}{8} \geq \frac{\hat{W}}{8 \log^5 N}$, and $w_0(B') \geq \frac{|B|}{8} \geq \frac{\hat{W}}{8 \log^5 N}$.

Recall that $d' \leq \frac{2d}{(\log N)^{28}} \leq \frac{n}{256} \leq \frac{\hat{W}}{128} \leq \frac{\sum_{e \in E(G_0)} \ell(e)}{2}$. Therefore, the length of every edge in G_0 is bounded by $\frac{\sum_{e \in E(G_0)} \ell(e)}{2}$. We call now apply the algorithm from Observation 4.5 to graph G_0, parameter $\frac{\varphi}{8}$, and the cut (A', B') in G_0, to compute a cut (A'', B'') in G_0 with $w_0(A'') \geq \frac{w_0(A')}{4} \geq \frac{\hat{W}}{32 \log^5 N}$ and $w_0(B'') \geq \frac{w_0(B')}{4} \geq \frac{\hat{W}}{32 \log^5 N}$, such that (A'', B'') is a strongly well-structured cut in G_0, whose sparsity is at most $\varphi/4$. The running time of the algorithm is bounded by $O(|E(G_0)|) \leq O(\hat{W}^2)$.

Lastly, we define the cut (A^*, B^*) in the current graph G as follows: $A^* = A'' \cap V(G)$ and $B^* = B'' \cap V(G)$. Since (A'', B'') is a strongly well-structured cut in G_0, and $G \subseteq G_0$, it is easy to verify that (A^*, B^*) is a strongly well-structured cut in G. Moreover, since the current phase has not terminated yet, the total weight of all vertices deleted so far from G is bounded by $\frac{\hat{W}}{\log^{30} N}$, so $w(A^*) \geq w_0(A^*) \geq w_0(A'') - \frac{\hat{W}}{\log^{30} N} \geq \frac{w_0(A'')}{2} > \frac{\hat{W}}{\log^{30} N}$. Similarly, $w(B^*) \geq w_0(B^*) \geq \frac{w_0(B'')}{2} > \frac{\hat{W}}{\log^{30} N}$. Since $|E_G(A^*, B^*)| \leq E_{G_0}(A'', B'')$, we conclude that cut (A^*, B^*) has sparsity at most φ. We output the cut (A^*, B^*) as the outcome of the algorithm. From the above discussion, the running time of the algorithm is bounded by $O(\hat{W}^2)$.

7.4 First Data Structure: Expander X and its Embedding

Throughout the phase, we will maintain an expander X, that is defined over the terminals, and is embedded into H. We start by providing the algorithm to initialize this data structure, and then provide the algorithm for maintaining it throughout the phase.

7.4.1 Initialization

In this step, we will either compute an expander X over the set T of vertices and embed it into H via short paths that cause low congestion; or we will compute a sparse cut (A, B) in H, following which a sufficient number of vertices will be deleted from H, so that the current phase will terminate. We summarize the algorithm for initializing the expander in the following lemma, whose statement uses the constant α_0 from Theorem 2.1.

Lemma 7.11 *There is a randomized algorithm, that either returns "FAIL", or computes one of the following:*

- *either a finalizing cut in H;*
- *or a strongly well-structured $1/d'$-sparse cut (A, B) in G with $w(A), w(B) \geq \frac{\hat{W}}{(\log N)^{30}}$;*
- *or a graph X^* with $V(X^*) = T$, such that X^* has maximum vertex degree at most 24, and it is an $\frac{\alpha_0}{81}$-expander. In the latter case, the algorithm additionally computes a set $F \subseteq E(X^*)$ of at most $\frac{\alpha_0 \cdot |T|}{500}$ edges, and an embedding $\mathcal{P}^*$ of $X^* \setminus F$ into H via paths that have length at most $O\left(d' \cdot (\log N)^{16}\right)$ each, that cause congestion $O\left((\log N)^{13}\right)$.*

The probability that the algorithm returns "FAIL" is at most $\frac{1}{N^7}$, and the running time of the algorithm is bounded by:

$$O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+3}\right).$$

The remainder of this subsubsection is dedicated to proving Lemma 7.11.

Recap of the Cut-Matching Game

We use the Cut-Matching Game – a framework that was initially proposed by [KRV09]. The specific variant of the game is due to [KKOV07]. Its slight modification (that we also use) was proposed in [CGL$^+$20], and later adapted to directed graphs in [BGS20]; the same modification was used in [CK24]. We also note that [Lou10] extended the original cut-matching game of [KRV09] to directed graphs, though we do not use this variation here. We now define the variant of the game that we use.

The game is played between two players: the Cut Player and the Matching Player. The game starts with a directed graph X that contains $\hat{n}$ vertices and no edges, and proceeds in iterations. In the ith iteration, the Cut Player chooses a cut (A_i, B_i) in X with $|A_i|, |B_i| \geq \hat{n}/10$, and $|E_X(A_i, B_i)| \leq \hat{n}/100$. It then computes two arbitrary equal-cardinality sets A'_i, B'_i of vertices with $|A'_i| \geq \hat{n}/4$, such that either $A_i \subseteq A'_i$, or $B_i \subseteq B'_i$ holds. The matching player then needs to return two perfect matchings: a matching $\mathcal{M}_i \subseteq A'_i \times B'_i$, and a matching $\mathcal{M}'_i \subseteq B'_i \times A'_i$, both of cardinality $|A'_i|$. The edges of $\mathcal{M}_i \cup \mathcal{M}'_i$ are added to X, and the iteration terminates. If there is no cut (A, B) in X with $|A|, |B| \geq \hat{n}/10$, and $|E_X(A, B)| \leq \hat{n}/100$, then the game must terminate, though we may choose to terminate it earlier. The following bound on the number of iterations in this variant of the Cut-Matching Game was shown by [BGS20], extending a similar result of [KKOV07] for undirected graphs.

Theorem 7.12 (Lemma 7.4 in [BGS20]; see also Theorem 7.3 and the following discussion) *The number of iterations in the above Cut-Matching Game is bounded by $O(\log \hat{n})$.*

From the above theorem, there is an absolute constant $c_{\mathsf{CMG}} > 0$, such that, after $c_{\mathsf{CMG}} \log \hat{n}$ iterations of the above game, for every cut (A, B) in X with $|A|, |B| \geq \hat{n}/10$, $|E_X(A_i, B_i)| > \hat{n}/100$ must hold. Notice that the maximum vertex degree in the final graph X is bounded by $2c_{\mathsf{CMG}} \log \hat{n}$.

Implementing the Cut Player

There are several known algorithms that we could use in order to implement the cut player. For example, [BGS20], generalizing the results of [CGL$^+$20] to directed graphs, provided such an algorithm, whose running time is $\widehat{O}(\hat{n})$ (see Theorem 7.3 in [BGS20]). We use instead the algorithm of [CK24], whose analysis is very simple, even though its running time is somewhat higher. The algorithm is summarized in the following theorem, that uses the constant α_0 from Theorem 2.1.

Theorem 7.13 (Theorem 8.5 in [CK24]) *There is a deterministic algorithm, that, given an $\hat{n}$-vertex directed graph G with maximum vertex degree at most $O(\log \hat{n})$, returns one of the following:*

- *Either a cut (A, B) in G with $|E_G(A, B)| \leq \hat{n}/100$ and $|A|, |B| \geq \hat{n}/10$; or*
- *another directed graph X' with $|V(X')| \geq \hat{n}/4$ and maximum vertex degree at most 18, such that X' is an α_0-expander, together with a subset $F \subseteq E(X')$ of at most $\frac{\alpha_0 \cdot \hat{n}}{1000}$ edges of X', and an embedding $\mathcal{P}$ of $X' \setminus F$ into G via paths of length at most $O(\log^2 \hat{n})$, that cause congestion at most $\tilde{\eta} \leq O(\log^2 \hat{n})$.*

The running time of the algorithm is $\widetilde{O}(\hat{n}^2)$.

Next, we discuss the implementation of the matching player, which is different from that of [CK24]: to make the algorithm more efficient, we rely on Algorithm $\mathcal{A}$ for the r-restricted RouteAndCut problem, whose existence we have assumed.

Implementing the Matching Player

We use the parameter $N' = \max\left\{N, \hat{W}\right\}$, so that $|V(H)| \leq N'$ and $N \leq N' \leq 12N$ holds, and additional parameters $\hat{\Delta} = \left\lceil \frac{|T|}{4} \right\rceil$ and $\eta' = (\log N')^8$. Throughout the algorithm, we will sometimes compute disjoint subsets A, B of vertices of H of cardinality $\hat{\Delta}$ each, and we will view $(H, A, B, \hat{\Delta}, \eta', N')$ as an instance of the r-restricted RouteAndCut problem. We show that this must be indeed a valid instance of the problem, in the following simple observation whose proof is deferred to Section D.3 of Appendix.

Observation 7.14 *For any pair (A, B) of disjoint subsets of $V(H)$ of cardinality $\hat{\Delta}$ each, $(H, A, B, \hat{\Delta}, \eta', N')$ is a valid input to the r-restricted* RouteAndCut *problem.*

The matching player will be implemented by applying Algorithm $\mathcal{A}$ for the r-restricted RouteAndCut problem to instances $(H, A, B, \hat{\Delta}, \eta', N')$, where A, B are disjoint subsets of $V(H)$ of cardinality $\hat{\Delta}$ each. Throughout, we will use a parameter $z = \frac{4|T|}{\log^4 N}$.

Proof of Lemma 7.11: the Cut-Matching Game

We let $\tilde{c}$ be a large enough constant, so that, in Theorem 7.13 the parameter $\tilde{\eta} \leq \tilde{c} \log^2 \hat{n}$. We start with the graph X, whose vertex set is T, and edge set is $\emptyset$, and then run the Cut-Matching game on it. Recall that the number of iterations in the game is bounded by $c_{\mathsf{CMG}} \cdot \log \hat{n}$. This ensures that the graph X that is maintained over the course of the game has maximum vertex degree $O(\log \hat{n})$.

Description of a Single Iteration

We now describe the execution of the ith iteration. First, we apply the algorithm from Theorem 7.13 to the current graph X. We say that the current iteration is a *type-1 iteration*, if the algorithm returns a cut (A_i, B_i) in X with $|E_X(A_i, B_i)| \leq \hat{n}/100$ and $|A_i|, |B_i| \geq \hat{n}/10$, and otherwise we say that it is a type-2 iteration. Assume first that the current iteration is a type-1 iteration. Then we compute two arbitrary disjoint subsets A'_i, B'_i of $V(X)$ of cardinality $\left\lceil \frac{\hat{n}}{4} \right\rceil = \left\lceil \frac{|T|}{4} \right\rceil = \hat{\Delta}$ each, such that either $A_i \subseteq A'_i$ or $B_i \subseteq B'_i$ holds. We treat (A'_i, B'_i) as the move of the cut player. Recall that the running time of the algorithm from Theorem 7.13 is $\widetilde{O}(\hat{n}^2) \leq \widetilde{O}(\hat{W}^2)$.

Next, we consider two instances of the r-restricted RouteAndCut problem: instance $(H, A'_i, B'_i, \hat{\Delta}, \eta', N')$, and instance $(H, B'_i, A'_i, \hat{\Delta}, \eta', N')$. We apply Algorithm $\mathcal{A}$ to each of these two instances. If the algorithm returns "FAIL" on any of these instances, we repeat it up to $\lceil 10 \log N \rceil$ times, or until it terminates with a valid output, whichever happens first. If, for any of the two instances of the RouteAndCut problem, the algorithm returned "FAIL" repeatedly in each of its $\lceil 10 \log N \rceil$ applications, then we terminate the algorithm and return "FAIL". Since the probability that an algorithm for the RouteAndCut problem returns "FAIL" is at most 1/2, the probability that the current iteration termintates with a "FAIL" is bounded by $\frac{2}{N^{10}}$. We assume from now on that the last application of Algorithm $\mathcal{A}$ to each of the two instances did not terminate with a "FAIL". We denote by $\mathcal{Q}'_i$ and

$\mathcal{Q}''_i$ the routings that these algorithms return, where $\mathcal{Q}'_i$ is the routing from A'_i to B'_i. Recall that the running time of Algorithm $\mathcal{A}$ on each of these two instances is bounded by:

$$\begin{aligned}
c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N'}} \cdot (\log N')^{16c_2(r-1)+8c_2} \\
&\leq O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log(12N))^{16c_2(r-1)+8c_2+1}\right) \\
&\leq O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+1} \cdot \left(1+\frac{1}{32c_2r}\right)^{16c_2(r-1)+8c_2+1}\right) \\
&\leq O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+1} \cdot \left(1+\frac{1}{32c_2r}\right)^{32c_2r}\right) \\
&\leq O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+1}\right)
\end{aligned}$$

(we have used Inequalities 17 and 18). Since Algorithm $\mathcal{A}$ may be applied up to $O(\log N)$ times to the resulting instances of the RouteAndCut problem, the total time spent in a single iteration by the executions of Algorithm $\mathcal{A}$ is bounded by:

$$O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+2}\right).$$

We now consider two cases. The first case happens when both $|\mathcal{Q}'_i|, |\mathcal{Q}''_i| \geq \hat{\Delta} - z$. In this case, we say that the current iteration is of *type (1a)*. Recall that the paths in each of the sets $\mathcal{Q}'_i$, $\mathcal{Q}''_i$ cause congestion at most $4\eta' \cdot \log N' \leq 4(\log N')^9 \leq 4(\log(12N))^9 \leq (\log N)^{10}$, since $\eta' = (\log N')^8$ and $N' \leq 12N$. Since regular and special edges must alternate on each such path, and since the total number of special edges is at most $\hat{W}/2$, $\sum_{P \in \mathcal{Q}'_i} |E(P)| \leq \hat{W} \cdot (\log N)^{10}$. Similarly, $\sum_{P \in \mathcal{Q}''_i} |E(P)| \leq \hat{W} \cdot (\log N)^{10}$. We say that a path $Q \in \mathcal{Q}'_i \cup \mathcal{Q}''_i$ is *long*, if it contains more than $8d' \cdot (\log N)^{14}$ edges. Clearly, the number of long paths in $\mathcal{Q}'_i$ is bounded by:

$$\frac{\hat{W} \cdot (\log N)^{10}}{8d' \cdot (\log N)^{14}} \leq \frac{\hat{W}}{2d'(\log N)^4} \leq \frac{|T|}{\log^4 N} \leq z.$$

(We have used the fact that $|T| \geq \frac{\hat{W}}{2d'}$, as Event $\mathcal{E}_1$ did not happen.) Similarly, at most z paths in $\mathcal{Q}''_i$ are long. We discard all long paths from both $\mathcal{Q}'_i$ and $\mathcal{Q}''_i$. At the end of this process, every path in $\mathcal{Q}'_i \cup \mathcal{Q}''_i$ has length at most $8d' \cdot (\log N)^{14}$, and $|\mathcal{Q}'_i|, |\mathcal{Q}''_i| \geq \hat{\Delta} - 2z$.

The set $\mathcal{Q}'_i$ of paths naturally defines a matching $\mathcal{M}'_i \subseteq A'_i \times B'_i$, where for every path $Q \in \mathcal{Q}_i$ whose endpoints are $a \in A'_i, b \in B'_i$, we add an edge $e = (a, b)$ to the matching. We also let $Q(e) = Q$ be its embedding into G. Similarly, the set $\mathcal{Q}''_i$ of paths defines a matching $\mathcal{M}''_i \subseteq B'_i \times A'_i$, together with an embedding $Q(e) \in \mathcal{Q}''_i$ for each edge $e \in \mathcal{M}''$. Recall that matching $\mathcal{M}'_i$ has cardinality at least $\hat{\Delta} - 2z$, while $|A'_i| = |B'_i| = \hat{\Delta}$. Therefore, by adding a set F'_i of at most $2z$ additional edges (that we refer to as *fake edges*), we can turn matching $\mathcal{M}'_i$ into a perfect one. Similarly, we add a collection F''_i of at most $2z$ fake edges to $\mathcal{M}''_i$ in order to turn it into a perfect matching. The edges of $\mathcal{M}'_i \cup \mathcal{M}''_i$ are then added to graph X, and the current iteration terminates.

Assume now that one of the two sets $\mathcal{Q}'_i, \mathcal{Q}''_i$ computed by Algorithm $\mathcal{A}$ contains fewer than $\hat{\Delta} - z$ paths; we assume w.l.o.g. that it is $\mathcal{Q}'_i$. In this case we say that the current iteration is of *type (1b)*.

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 $\mathcal{Q}'_i$, so $|A''_i|, |B''_i| \geq z$. Recall that the algorithm must also return a cut (Z, Z') in H, such that:

$$|E_H(Z,Z')| \leq \frac{64\hat{\Delta}}{\eta' \log^4 \hat{W}} + \frac{256|\mathcal{Q}'_i|}{\eta'} \leq \frac{512\hat{\Delta}}{\eta'} \leq \frac{256|T|}{(\log N')^8} \leq \frac{\hat{W}}{d'(\log N)^7},$$

since $\eta' = (\log N')^8$, $\hat{\Delta} = \left\lceil \frac{|T|}{4} \right\rceil$, $|T| \leq \frac{8\hat{W}}{d'}$ since Event $\mathcal{E}_1$ did not happen, and $N' \geq N$. We are also guaranteed that $A''_i \subseteq Z$ and $B''_i \subseteq Z'$, so $|Z \cap T|, |Z' \cap T| \geq z = \frac{4|T|}{\log^4 N}$. We use the following simple claim, that either computes a finalizing cut in H, or returns a strongly well-structured $1/d'$-sparse cut (A,B) in G with $w(A), w(B) \geq \frac{\hat{W}}{(\log N)^{30}}$. The proof is deferred to Section D.4 of Appendix.

Claim 7.15 *There is a deterministic algorithm, that, given a cut (Z,Z') in graph H with $|Z \cap T|, |Z' \cap T| \geq \frac{4|T|}{\log^4 N}$ and $|E_H(Z,Z')| \leq \frac{\hat{W}}{d'(\log N)^7}$, either computes a finalizing cut in H, or returns a strongly well-structured $1/d'$-sparse cut (A,B) in G with $w(A), w(B) \geq \frac{\hat{W}}{(\log N)^{30}}$. The running time of the algorithm is $O(\hat{W}^2)$.*

If the algorithm from Claim 7.15 returns a finalizing cut in H, then we output this cut as the outcome of the algorithm. Otherwise, we output the $1/d'$-sparse cut (A,B) in G with $w(A), w(B) \geq \frac{\hat{W}}{(\log N)^{30}}$ that the algorithm returns.

Finally, assume that the current iteration i is of type 2. In this case, the algorithm for the Cut Player from Theorem 7.13 computed a directed graph X' with $|V(X')| \geq \hat{n}/4$ and maximum vertex degree at most 18, such that X' is an α_0-expander, together with a subset $F' \subseteq E(X')$ of at most $\frac{\alpha_0 \cdot n}{1000}$ edges of X', and an embedding $\mathcal{P}$ of $X' \setminus F'$ into X via paths of length at most $O(\log^2 \hat{n})$, that cause congestion at most $\tilde{c} \log^2 \hat{n}$.

Denote $\hat{F} = \bigcup_{i'=1}^{i-1}(F'_i \cup F''_i)$ – the collection of the fake edges that we have computed so far. Since for all $1 \leq i' < i$, $|F'_i \cup F''_i| \leq 4z$ and $i \leq c_{\mathsf{CMG}} \log n$, we get that $|\hat{F}| \leq 4c_{\mathsf{CMG}} z \log \hat{n}$.

We also denote $\hat{\mathcal{Q}} = \bigcup_{i'=1}^{i-1}(\mathcal{Q}'_i \cup \mathcal{Q}''_i)$. Note that $\hat{\mathcal{Q}}$ defines an embedding of $X \setminus \hat{F}$ into H. All paths in $\hat{\mathcal{Q}}$ have lengths at most $8d' \cdot (\log N)^{14}$. Each set $\mathcal{Q}'_i \cup \mathcal{Q}''_i$ causes congestion at most $2(\log N)^{10}$, and so the total congestion caused by the paths in $\hat{\mathcal{Q}}$ is bounded by $\hat{\eta} = O\left((\log N)^{11}\right)$. We now terminate the execution of the Cut-Matching Game.

As our next step, we will construct a set $F \subseteq E(X')$ of fake edges, and an embedding of the remaining edges of X' into H. The latter is done by comining the embedding $\mathcal{P}$ of $X' \setminus F'$ into X with the embedding $\hat{\mathcal{Q}}$ of $X \setminus \hat{F}$ into H.

Constructing the Set F of Fake Edges for X' and Embedding $X' \setminus F$ into H

We start with $F = F'$. Next, we process every edge $e \in E(X') \setminus F'$ one by one. When an edge $e = (u,v)$ is processed, we consider the path $P'(e) \in \mathcal{P}$, that serves as an embedding of e into graph X. Recall that the length of the path is at most $O(\log^2 \hat{n})$. Let $P(e) = (e_1, e_2, \ldots, e_k)$, where $k \leq O(\log^2 \hat{n}) \leq O(\log^2 N)$. If path $P(e)$ contains any edges of $\hat{F}$, then we add e to the set F of fake edges. Otherwise, for all $1 \leq j \leq k$, we consider the path $Q(e_j) \in \hat{\mathcal{Q}}$, that serves as the embedding of the edge e_j into graph G. Recall that the length of Q is at most $8d' \cdot (\log N)^{14}$. By concatenating the paths $Q(e_1), \ldots, Q(e_k)$, we obtain a path $P'(e)$ connecting u to v in G, whose length is at most $O\left(d' \cdot (\log N)^{16}\right)$. We turn $P'(e)$ into a simple path in time $\tilde{O}(d')$, obtaining the final path $\hat{P}(e)$, that will serve as an embedding of the edge e.

Once every edge of $X' \setminus F'$ is processed, we obtain a final set F of fake edges, and, for every remaining edge $e \in E(X') \setminus F$, a simple path $\hat{P}(e)$ of length $O\left(d' \cdot (\log N)^{16}\right)$, that serves as the embedding of

e in G. Recall that $|V(X')| \geq \hat{n}/4$, and graph X' is an α_0-expander with maximum vertex degree at most 18. Next, we bound the congestion caused by the paths in set $\mathcal{P}^* = \left\{\hat{P}(e) \mid e \in E(X') \setminus F\right\}$ in H, and the cardinality of the set F of fake edges.

Consider any edge $e \in E(H)$. Recall that e may participate in at most $\hat{\eta} = O\left((\log N)^{11}\right)$ paths in set $\hat{\mathcal{Q}} = \left\{Q(e') \mid e' \in E(X) \setminus \hat{F}\right\}$. Every edge $e' \in E(X) \setminus \hat{F}$, whose corresponding path $Q(e')$ contains the edge e, may in turn participate in at most $O(\log^2 N)$ paths of $\mathcal{P}$. Therefore, overall, every edge e of G participates in at most $O\left((\log N)^{13}\right)$ paths of $\mathcal{P}^*$.

In order to bound the number of the fake edges in F, recall that $|\hat{F}| \leq 4c_{\mathsf{CMG}} z \log N$. Since the paths in $\mathcal{P}$ cause congestion at most $\tilde{c} \log^2 \hat{n} = \tilde{c} \log^2 N$, we get that every edge of $\hat{F}$ may participate in at most $\tilde{c} \log^2 N$ paths of $\mathcal{P}$. Therefore, $|F \setminus \hat{F}| \leq (4c_{\mathsf{CMG}} z \log N) \cdot (\tilde{c} \log^2 N) \leq 4\tilde{c} \cdot c_{\mathsf{CMG}} \cdot z \cdot \log^3 N \leq \frac{\alpha_0 \cdot |T|}{2000}$, since $z = \frac{4|T|}{\log^4 N}$, and N is sufficiently large. Since $|F'| \leq \frac{\alpha_0 \cdot |T|}{1000}$, we get that $|F| \leq \frac{3\alpha_0 \cdot |T|}{2000}$.

Note that graph X' has all required properties, except that $V(X')$ may not contain all terminals. In our last step, we fix this problem, by connecting the terminals that do not lie in X' to the vertices of X'.

Extending the Expander to All Terminals

Let $T_0 \subseteq T$ be the set of terminals that lie in X', so $|T_0| \geq \left\lceil \frac{|T|}{4} \right\rceil$. We partition the terminals of $T \setminus T_0$ into three subsets T_1, T_2, T_3 arbitrarily, so that every subset contains at most $\left\lceil \frac{|T|}{4} \right\rceil$ terminals. We now perform last three iterations, that we will denote by iterations $(i+1), (i+2)$ and $(i+3)$. In each of these three iterations, we will only invoke the algorithm of the Matching Player, in order to connect the vertices in one of the subsets T_1, T_2, T_3 to T_0.

We now describe iteration $(i+1)$. The remaining two iterations are performed similarly. We let A'_{i+1} be any subset of T_0 containing exactly $\hat{\Delta} = \left\lceil \frac{|T|}{4} \right\rceil$ vertices. We also let B'_{i+1} be any subset of $\hat{\Delta}$ terminals with $T_1 \subseteq B'_{i+1}$, so that A'_{i+1} and B'_{i+1} are disjoint. Next, we consider two instances of the r-restricted $\mathsf{RouteAndCut}$ problem: instance $(H, A'_{i+1}, B'_{i+1}, \hat{\Delta}, \eta', N')$, and instance $(H, B'_{i+1}, A'_{i+1}, \hat{\Delta}, \eta', N')$. We apply Algorithm $\mathcal{A}$ to each of these two instances. As before, if either of these algorithms returns "FAIL", then we repeat it at most $\lceil 10 \log N \rceil$ times, or until it terminates with a valid input – whatever happens first. If the algorithm returned "FAIL" in all $\lceil 10 \log N \rceil$ applications, then we terminate the current algorithm with a "FAIL". We assume from now on that in the last application of algorithm $\mathcal{A}$ to both instances a valid output was computed, and we denote by $\mathcal{Q}'_{i+1}$ and $\mathcal{Q}''_{i+1}$ the routings that these algorithms return, where $\mathcal{Q}'_{i+1}$ is the routing from A'_{i+1} to B'_{i+1}.

If either of the sets $\mathcal{Q}'_{i+1}, \mathcal{Q}''_{i+1}$ of paths has cardinality less than $\hat{\Delta} - z$, then we employ the same algorithm as before to either compute a finalizing cut in H, or to compute a strongly well-structured $1/d'$-sparse cut (A, B) in G with $w(A), w(B) \geq \frac{\hat{W}}{(\log N)^{30}}$. We then terminate the algorithm and return the resulting cut. We assume from now on that $|\mathcal{Q}'_{i+1}|, |\mathcal{Q}''_{i+1}| \geq \hat{\Delta} - z$. We use the set $\mathcal{Q}'_{i+1}$ of paths as before (except that we ingore those paths that terminate at vertices of $B'_{i+1} \setminus T_1$), in order to compute a matching $M'_{i+1} \subseteq T_0 \times T_1$, with $|M'_{i+1}| \geq |T_1| - 2z$, together with an embedding of the edges of M'_{i+1} into the graph H, via paths of lengths at most $8d' \cdot (\log N)^{14}$, that cause congestion at most $O(\log^{10} N)$. We then add a collection F'_{i+1} of at most $2z$ fake edges to M'_{i+1} to turn it into a matching of cardinality $|T_1|$. Similarly, we use the paths in $\mathcal{Q}''_{i+1}$, in order to compute a matching $M''_{i+1} \subseteq T_1 \times T_0$, with $|M''_{i+1}| = |T_1|$, together with a subset $F''_{i+1} \subseteq M''_{i+1}$ of at most $2z$ fake edges, and an embedding of the edges of $M''_{i+1} \setminus F''_{i+1}$ into the graph H, via paths of lengths at most $8d' \cdot (\log N)^{14}$, that cause congestion at most $O(\log^{10} N)$. The remaining two iterations are executed similarly, except that we use sets T_2 and T_3 of terminals instead of T_1.

Consider now the final graph X^*, that is obtained from X' by adding the vertices of $T_1 \cup T_2 \cup T_3$ to it, and the edges of $\mathcal{M}^* = \bigcup_{j=1}^{3}(M'_{i+j} \cup M''_{i+j})$. Then $V(X^*) = T$, maximum vertex degree in X^* is at most $18 + 6 = 24$. Moreover, from Observation 2.2, graph X^* is an $\frac{\alpha_0}{81}$-expander.

Recall that each of the sets $\left\{F'_{i+j}, F''_{i+j} \mid 1 \le j \le 3\right\}$ of fake edges has cardinality at most $2z \le \frac{8|T|}{\log^4 N}$. We add the fake edges from all these sets to the set F of fake edges. Since initially $|F| \le \frac{3\alpha_0 \cdot |T|}{2000}$, and since the number of edges that we added to F in this step is bounded by $\frac{100|T|}{\log^4 N}$, we get that $|F| \le \frac{\alpha_0 \cdot |T|}{500}$ holds. For every edge $e \in \mathcal{M}^* \setminus F$, we have computed an embedding path $\tilde{P}(e)$, that is added to the set $\mathcal{P}^*$ of paths. The final set $\mathcal{P}^*$ of paths is an embedding of $X^* \setminus F$ into H, where the length of every path is $O\left(d' \cdot (\log N)^{15}\right)$, and the paths in $\mathcal{P}^*$ cause congestion $O\left((\log N)^{16}\right)$.

Running Time and Failure Probability Analysis

The algorithm has $O(\log \hat{n}) \le O(\log N)$ iterations. The probability that it returns "FAIL" in a single iteration is at most $\frac{2}{N^{10}}$, and so the probability that the algorithm ever returns "FAIL" is at most $\frac{1}{N^9}$. the total time spent in a single iteration on the executions of Algorithm $\mathcal{A}$ is bounded by:

$$O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+2}\right).$$

Additionally, in every iteration we use the algorithm for the Cut Player from Theorem 7.13, whose running time is $\widetilde{O}(\hat{W}^2)$.

The algorithm from Claim 7.15, whose running time is $O(\hat{W}^2)$, may only be incurred once over the course of the execution of our algorithm.

Additionally, when computing the embedding $\mathcal{P}^*$ of $X^* \setminus F$ into G, we spend time at most $O\left(d' \cdot (\log N)^{16}\right)$ per edge of X', and, since $|E(X')| \le O(\hat{W})$, the time required to compute the embedding is bounded by: $O\left(\hat{W} \cdot d' \cdot (\log N)^{15}\right) \le O\left(\hat{W}^2\right)$, since $d' \le \frac{2d}{(\log N)^{28}}$ and $d \le n$ from Inequality 15. Overall, the running time of the algorithm is bounded by:

$$\begin{aligned} O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+3}\right) + \tilde{O}(\hat{W}^2) \\ \le O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+3}\right). \end{aligned}$$

7.4.2 Maintaining the Expander

At the beginning of the phase, we execute the algorithm from Lemma 7.11. If the algorithm terminated with a "FAIL", which may only happen with probability at most $\frac{1}{N^7}$, then we terminate the algorithm for the current phase and return "FAIL". If the algorithm returned a finalizing cut in H, then we apply the algorithm from Claim 7.8 to this cut. If the algorithm from Claim 7.8 estabishes that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ happened, we terminate the phase with a "FAIL". Otherwise, we obtain a strongly well-structured φ-sparse cut (A^*, B^*) in the current graph G, such that $w(A^*), w(B^*) \ge \frac{\hat{W}}{\log^{30} N}$. In this case, we output the cut (A^*, B^*). Once this cut is processed, at least $\frac{\hat{W}}{\log^{30} N}$ vertices are deleted from H, and the current phase terminates. Finally, if the algorithm from Lemma 7.11 returns a strongly well-structured $1/d'$-sparse cut (A,B) in G with $w(A), w(B) \geq \frac{\hat{W}}{(\log N)^{30}}$, then as before we output this cut, and, once it is processed, at least $\frac{\hat{W}}{\log^{30} N}$ vertices are deleted from H, and the current phase terminates. Therefore, we assume from now on that the algorithm from Lemma 7.11 computed a graph X^*, that we will denote for simplicity by X from now on. Recall that $V(X) = T$, maximum vertex degree in X is at most 24, and X is an $\frac{\alpha_0}{81}$-expander. Recall that we are also given a set $F \subseteq E(X)$ of at most $\frac{\alpha_0 \cdot |T|}{500}$ fake edges, and an embedding $\mathcal{P}^*$ of $X \setminus F$ into H. We let $d'' = \Theta\left(d' \cdot (\log N)^{15}\right)$ and $\eta'' = \Theta\left((\log N)^{13}\right)$ be such that every path in $\mathcal{P}^*$ has length at most d'', and the congestion caused by the paths in $\mathcal{P}^*$ is at most η''. For convenience, we will denote the set $\mathcal{P}^*$ of paths by $\mathcal{Q}$ from now on, and, for every edge $\hat{e} \in E(X) \setminus F$, we denote by $Q(\hat{e}) \in \mathcal{Q}$ its corresponding embedding path.

Recall that, since Event $\mathcal{E}_1$ did not happen, we are guaranteed that $\frac{\hat{W}}{2d'} \leq |T| \leq \frac{8\hat{W}}{d'}$. As the phase progresses, some vertices and edges may be deleted from X and from G. We will view the set T of terminals as a static set, so even if some vertices of T are deleted from X or from G, the set T remains unchanged.

Throughout the algorithm, for every edge $e \in E(H)$, we maintain a collection $S'(e) \subseteq E(X)$ of edges $\hat{e} \in E(X) \setminus F$, such that $e \in Q(\hat{e})$. Clearly, the time required to compute these initial sets $S'(e)$ of edges, for all $e \in E(H)$, is subsumed by the time required to compute the embedding of $X \setminus F$ into H. Additionally, for every edge $\hat{e} \in E(X) \setminus F$, we maintain the length $\hat{\ell}(\hat{e})$ of the path $Q(\hat{e})$, which is equal to the number of edges on the path, and must initially be bounded by d''.

As the phase progresses, graph H undergoes changes, that may require updating the expander X and its embedding. Specifically, we need to ensure that, even as vertices and edges are deleted from H, $V(X) \subseteq V(H)$, and, for every edge $\hat{e} \in E(X) \setminus F$, $Q(\hat{e}) \subseteq H$ continue to hold. Additionally, as edges in H undergo splitting, some of the embedding paths in $\mathcal{Q}$ may become longer, and we may need to delete the corresponding edges from X, in order to ensure that the lengths of all embedding paths are suitably bounded. Lastly, we need to ensure that, throughout the phase, for every pair $x, y \in V(X)$ of vertices, there is a path connecting x to y in $X \setminus F$, whose length is at most $O(\log N)$. Whenever our algorithm discovers that this is no longer the case, it may compute a sparse cut (A,B) in X, and delete the vertices of the smaller side of the cut from X. We now describe all updates that graph X undergoes over the course of the phase formally. We will ensure that, despite these updates, throughout the phase, $|V(X)| \geq |T|/2$ holds.

First Type of Updates: Due to Edge Splitting

Recall that, as the algorithm progresses, the lengths of some edges $e \in E(G)$ may be doubled. Whenever this happens, we update the corresponding path $P^*(e)$ in H, by splitting every special edge $e' \in E(P^*(e))$. The latter is done by replacing the special edge $e' = (x,y)$ with a path (x, z, z', y), where z, z' are new vertices, and (x,z), (z', y) are new special edges. Notice that once edge e' is split, it no longer lies in graph H. For every edge $\hat{e} \in E(X) \setminus F$ that lies in $S'(e')$, we consider the corresponding embedding path $Q(\hat{e})$, that must contain the edge e'. We replace e' with the new path (x, z, z', y) on $Q(\hat{e})$, and insert $\hat{e}$ into the sets $S'((x,z))$, $S'((z',y))$, that correspond to the new special edges (x,z) and (z',y). We also update the length $\hat{\ell}(\hat{e})$ of path $Q(\hat{e})$, which increases by additive 2. If, as the result of this procedure, the length of path $Q(\hat{e})$ became greater than $8d''$, then edge $\hat{e}$ is deleted from graph X.

We denote by $\hat{E}_1$ the set of all edges that were deleted from graph X over the course of the phase due to this type of updates. We now show that the number of such edges must be small.

Claim 7.16 $|\hat{E}_1| \leq \frac{\alpha_0 \cdot |T|}{5000}$.

Proof: Recall that, over the course of the phase, the algorithm only needs to respond to at most Δ' queries. Given such a query between a pair (x, y) of vertices, the algorithm must return a path P of length at most d connecting x to y. Denote by $\tilde{\ell}(P)$ the length of the path P at the time when it was returned in response to the query. Then the lengths of some edges on path P are doubled. If $\mathcal{P}$ is the set of all paths that the algorithm returned in response to queries over the course of the phase, then $\sum_{P\in\mathcal{P}}\tilde{\ell}(P) \leq \Delta' \cdot d$ must hold.

Let G_0 denote the graph G at the beginning of the phase. Recall that, if e is any edge of G_0, and τ, τ' are two consecutive times during the current phase when the length of e is doubled, then e must have participated in at least η paths that the algorithm returned during the time interval $(\tau, \tau']$. However, the first time that the length of e is doubled during the current phase may occur even before it participates in η such paths. For every edge $e \in E(G_0)$, denote by $\tilde{\ell}^0(e)$ the length of e at the beginning of the current phase. If $e \in E(G)$ at the end of the phase, then we denote by $\tilde{\ell}^*(e)$ the length of e at the end of the phase; otherwise, $\tilde{\ell}^*(e)$ is the length of e just before it is deleted from G.

Let $E' \subseteq E(G_0)$ be the set of all edges e with $\tilde{\ell}^*(e) > 2\tilde{\ell}^0(e)$, so E' contains all edges of G_0 whose lengths were doubled at least twice. Consider now any edge $e \in E'$, and assume that $\tilde{\ell}^*(e) = 2^i$. Let τ be the time during the current phase when the length of e was doubled from 2^{i-2} to 2^{i-1}, and let τ' be the time when it was doubled from 2^{i-1} to 2^i. Then e participated in at least η paths that were returned by the algorithm in response to queries during the time interval $(\tau, \tau']$. For each such path P, it contributed $2^{i-1} = \frac{\tilde{\ell}^*(e)}{2}$ to $\tilde{\ell}(P)$. Therefore, every edge $e \in E'$ contributes at least $\frac{\eta\cdot\tilde{\ell}^*(e)}{2}$ to $\sum_{P\in\mathcal{P}}\tilde{\ell}(P)$. We conclude that $\sum_{e\in E'}\tilde{\ell}^*(e) \leq \frac{2}{\eta}\sum_{P\in\mathcal{P}}\tilde{\ell}(P) \leq \frac{2\Delta'\cdot d}{\eta}$.

Notice that, for every edge $e \in E(G)$, at any time during the execution of the algorithm, the number of special edges on $P^*(e)$ is equal to the current length of e, while the splitting of a speical edge on $P^*(e)$ increases the number of special edges on $P^*(e)$ by 1. Therefore, if the current length of e is $\ell(e)$, then the edges of $P^*(e)$ underwent $\ell(e) - \tilde{\ell}^0(e)$ splittings since the beginning of the algorithm.

Consider the graph H at the beginning of the current phase, that we denote by H_0. Note that, whenever a special edge $e \in E(H)$ undergoes a splitting, the edge itself is deleted from H, and instead two new edges are inserted. Consider now a special edge $e \in E(H)$, and suppose it underwent a splitting at some time. We say that this splitting operation is *unimportant* if $e \in E(H_0)$, and we say that it is *important* otherwise. Note that, if an edge $e \in E(H)$ undergoes an important splitting update, then its parent-edge must lie in E'. Therefore, from the above discussion, the total number of important splitting operations in graph H is bounded by $\sum_{e\in E'}\tilde{\ell}^*(e)$.

Consider now some edge $\hat{e} \in E(X) \setminus F$, and its embedding path $Q(\hat{e})$. Let $\mu(\hat{e})$ be the total number of important splitting operations that the edges of path $Q(\hat{e})$ underwent over the course of the algorithm. Since the embedding $\mathcal{Q}$ of $X \setminus F$ into H causes congestion at most η'', we get that $\sum_{e\in X\setminus F}\mu(\hat{e}) \leq \eta'' \cdot \sum_{e\in E'}\tilde{\ell}^*(e) \leq \frac{2\Delta'\cdot d\cdot\eta''}{\eta}$. On the other hand, if $\hat{e} \in \hat{E}_1$, then the length of $Q(\hat{e})$ increased from d'' to $8d''$ over the course of the algorithm, and so the edges on $Q(\hat{e})$ must have undergone at least $6d''$ important splitting operations so $\mu(\hat{e}) \geq 6d''$. Therefore:

$$|\hat{E}'| \leq \frac{2\Delta'\cdot d\cdot\eta''}{6\eta d''} \leq O\left(\frac{\Delta\cdot d}{\eta\cdot d'\cdot(\log N)^{33}}\right) \leq O\left(\frac{n}{d'\cdot(\log N)^{33}}\right) \leq O\left(\frac{\hat{W}}{d'\cdot(\log N)^{33}}\right) < \frac{\alpha_0\cdot|T|}{5000},$$

since $d'' = \Theta\left(d'\cdot(\log N)^{16}\right)$, $\Delta' = \frac{\Delta}{(\log N)^{30}}$, and $\eta'' = \Theta\left((\log N)^{13}\right)$. We have also used the fact that $\frac{\Delta\cdot d}{\eta} \leq n$ from the problem definition, that $\hat{W} \geq \frac{n}{2}$, and that $|T| > \frac{8\hat{W}}{d'}$, since Event $\mathcal{E}_1$ did not happen. $\square$

Second Type of Updates: Due to Sparse Cuts in X

Throughout the execution of the phase, we may sometimes find cuts (A, B) in $X \setminus F$, with $|E_{X\setminus F}(A, B)| \leq \frac{\alpha_0}{2^{20}} \cdot \min\{|A|, |B|\}$. Denote A and B by Z and Z', where Z is the smaller of the two sets, breaking ties arbitrarily. We then delete the vertices of Z from graph X. Notice that $|E_{X\setminus F}(A, B)| \leq \frac{\alpha_0}{2^{20}} \cdot |Z|$. We *charge* the vertices of Z for the edges of $E_{X\setminus F}(A, B)$, where each vertex $v \in Z$ is charged $\frac{\alpha_0}{2^{20}}$ units, so the total charge is at least $|E_{X\setminus F}(A, B)|$.

Let $\hat{E}_2$ be the set of edges in the initial graph X, that is defined as follows. Initially, $\hat{E}_2 = \emptyset$. Whenever our algorithm computes a cut (A, B) in $X \setminus F$, with $|E_{X\setminus F}(A, B)| \leq \frac{\alpha_0}{2^{20}} \cdot \min\{|A|, |B|\}$, we add the edges of $E_{X\setminus F}(A, B)$ to set $\hat{E}_2$. From the charging scheme that we just described, it is easy to verify that, at the end of the algorithm, $|\hat{E}_2| \leq \frac{\alpha_0 \cdot |T|}{2^{20}}$ holds.

Third Type of Updates: Due to Sparse Cuts in G

Recall that our algorithm may, at any time, produce a strongly well-structured φ-sparse cut (A, B) in graph G. If $w(A) \leq W(B)$, then the vertices of A are deleted from G, and otherwise the vertices of B are deleted from G. Additionally, the endpoints of the edges of $E_G(A, B)$ are also deleted from G, and graph H is also updated with corresponding edge- and vertex-deletions. This may require updating graph X, for example, if some of the embedding paths in $\mathcal{Q}$ use vertices or edges that were deleted from H. We emphasize that edges and vertices may only be deleted from G via this procedure, and the only other updates that graph G undergoes is the doubling of the lengths of its edges.

Let $(A_1, B_1), \ldots, (A_q, B_q)$ denote the sequence of all strongly well-structured φ-sparse cuts in G that our algorithm produced during the current phase, indexed in the order in which the algorithm produced them. If the last sparse cut that the algorithm produced caused the total number of vertices that were deleted from H over the course of the phase to reach $\frac{\hat{W}}{\log^{30} N}$ (after which the current phase is immediately terminated), then we do not include that cut in this sequence. For all $1 \leq i \leq q$, denote by τ_i the time when cut (A_i, B_i) was produced, by $E_i = E_G(A_i, B_i)$ at time τ_i, and by $J_i \subseteq A_i, J'_i \subseteq B_i$ the vertices that serve as endpoints of the edges in E_i (with respect to $G^{(\tau_i)}$ – graph G at time τ_i). If $w(A_i) \geq w(B_i)$, then we denote $Z_i = A_i \setminus J_i$, and otherwise we denote $Z_i = B_i \setminus J'_i$. We also let $\overline{Z}_i = V(G^{(\tau_i)}) \setminus Z_i$ contain the remaining vertices of the current graph $G^{(\tau_i)}$. For convenience, if $Z_i = A_i \setminus J_i$, then we say that cut (A_i, B_i) is of *type 1*, and otherwise we say that it is of *type 2*.

We denote by G_0 the graph G at the beginning of the phase, and, for all $1 \leq i \leq q$, we denote by G_i the graph G that is obtained after cut (A_i, B_i) has been processed. Therefore, for all $1 \leq i \leq q$, graph G_i is obtained from graph G_{i-1} by deleting the vertices of $\overline{Z}_i$ from it, or, equivalently, $G_i = G_{i-1}[Z_i]$.

We denote by H_0 the graph H at the beginning of the current phase. For all $1 \leq i \leq q$, we denote by H'_i the graph H just before the cut (A_i, B_i) is processed, and we denote by H_i the graph H immediately after cut (A_i, B_i) is processed. Therefore, graph H_i is obtained from H'_i by deleting from it all vertices and regular edges that were deleted from G_{i-1}; additionally, for every special edge e that was deleted from G_{i-1}, all vertices and edges on the corresponding path $P^*(e)$ are deleted from H'_i. We denote $V(H_i)$ by Z^*_i, and we denote by $\overline{Z}^*_i = V(H'_i) \setminus V(H_i)$ the set of all vertices that were deleted from H'_i when cut (A_i, B_i) of G was processed.

We note that, in addition to the updates due to processing the cuts (A_j, B_j), the only other kind of updates that graph H undergoes is the splitting of some of its special edges, that happens whenever the length of the corresponding parent-edge is doubled. Therefore, for all $1 \leq i \leq q$, graph H'_i can be obtained from H_{i-1} by executing a sequence of special edge splitting operations.

Consider some index $1 \leq i \leq q$, and consider an edge $e = (x, y) \in E_i$. Recall that this edge is represented by a path $P^*(e)$ in graph H. Additionally, every regular edge e' that is incident to x or y in G is present in graph H. We let $\Gamma(e) \subseteq E(H'_i)$ contain all regular edges that are incident to x or y in H'_i, and all edges of $P^*(e)$. We then set $\Gamma_i = \bigcup_{e \in E_i} \Gamma(e)$. Notice that, if (A_i, B_i) is a type-1 cut (so $Z_i = A_i \setminus J_i$), then graph $H'_i \setminus \Gamma_i$ contains no edge connecting a vertex of Z^*_i to a vertex of $\overline{Z}^*_i$. Similarly, if (A_i, B_i) is a type-2 cut (so $Z_i = B_i \setminus J'_i$), the graph $H'_i \setminus \Gamma_i$ contains no edge connecting a vertex of $\overline{Z}^*_i$ to a vertex of Z^*_i. Therefore, it may be convenient to think of the update to graph H'_i due to the cut (A_i, B_i) in G as follows: first, we delete the edges of Γ_i from H'_i. Then we obtain a sparsity-0 cut in the resulting graph (with the cut being $(Z^*_i, \overline{Z}^*_i)$ if (A_i, B_i) is a type-1 cut, or $(\overline{Z}^*_i, Z^*_i)$ otherwise). After that we delete the vertices of one of the sides of the cut (vertices of $\overline{Z}^*_i$) from the resulting graph, obtaining graph H_i.

Before we describe the updates to the expander X, recall that X undergoes other types of updates, in which edges and vertices may be deleted from it. But for the sake of analyzing the updates of type 3, it will be helpful for us to ignore the other types of updates that expander X undergoes. Therefore, we denote the initial graph X by X_0, and, for $1 \leq i \leq q$, we denote by X_i the graph that is obtained from X_{i-1} once the cut (A_i, B_i) is processed. Except for updating the graph X while processing the cuts (A_i, B_i), we currently do not allow any other updates to X. We will show how combine all types of updates to X together later.

Fix again an index $1 \leq i \leq q$, and consider an edge $e = (x, y) \in E_i$. Recall that we have defined a collection $\Gamma(e)$ of edges of H'_i, that includes all edges on the path $P^*(e)$, and all regular edges that are incident to x or to y in H'_i. We define the corresponding subset $\hat{\Gamma}(e)$ of edges of $X_{i-1} \setminus F$: $\hat{\Gamma}(e) = \bigcup_{e' \in \Gamma(e)} S'(e')$. In other words, an edge $\hat{e} \in E(X_{i-1}) \setminus F$ belongs to the set $\hat{\Gamma}(e)$ if and only if its corresponding embedding path $Q(\hat{e})$ contains at least one edge of $\Gamma(e)$. We need the following simple observation.

Observation 7.17 *For every edge $e = (x, y) \in E_i$, $|\hat{\Gamma}(e)| \leq 2\eta''$.*

Proof: Let $T(e)$ be the set of all terminals that lie on path $P^*(e)$. Since we have assumed that Event $\mathcal{E}_4$ did not happen, $|T(e)| \leq 16 \log N$.

We classify the edges in set $\hat{\Gamma}(e)$ into three different types, and bound the number of edges of each type separately. Consider any edge $\hat{e} \in \hat{\Gamma}(e)$.

We say that $\hat{e}$ is of type 1, if at least one of its endpoints lies on $P^*(e)$. On this case, at least one endpoint of $\hat{e}$ belongs to set $T(e)$. Since $|T(e)| \leq 16 \log N$, and since the degree of every vertex in X is at most 24, we get that the total number of type-1 edges in $\hat{\Gamma}(e)$ is bounded by $O(\log N) \leq \eta''$.

We say that edge $\hat{e}$ is of type 2, if it is not of type 1, and, additionally, its embedding path $Q(\hat{e})$ contains at least one edge of $P^*(e)$. Since every inner vertex on $P^*(e)$ has in- and out-degree 1, and the endpoints of $\hat{e}$ do not lie on $P^*(e)$, it must be the case that $P^*(e) \subseteq Q(\hat{e})$. Since the embedding of $X \setminus F$ into H causes congestion at most η'', we get that the total number of type-2 edges in $\hat{\Gamma}$ is bounded by η''.

If edge $\hat{e}$ is not of the first two types, then we say that it is of type 3. We claim that no type-3 edges may exist. Indeed, assume for contradiction that some edge $\hat{e} \in \hat{\Gamma}(e)$ is of type 3. Then path $Q(\hat{e})$ must contain some regular edge e' of H'_i that is incident to x or y; assume w.l.o.g. that e' is incident to x. However, x is not an endpoint of $\hat{e}$, and so it is not an endpoint of $Q(\hat{e})$. Since the only edge leaving x in G is e, path $Q(\hat{e})$ must contain the first edge on $P^*(e)$. But then $\hat{e}$ must belong to one of the first two types, a contradiction. □

Fix again an index $1 \leq i \leq q$. We now define the set $\hat{\Gamma}_i$ of edges of $X_{i-1} \setminus F$, as follows: $\hat{\Gamma}_i = \bigcup_{e \in E_i} \hat{\Gamma}(e)$. Note that, equivalently, $\hat{\Gamma}_i$ contains all edges $\hat{e}$ of $X_{i-1} \setminus F$, whose embedding path $Q(\hat{e})$ contains at least one edge of Γ_i. Notice that, from Observation 7.17, $|\hat{\Gamma}_i| \le 2|E_i| \cdot \eta''$. We denote $\hat{E}_3 = \bigcup_{i=1}^q \hat{\Gamma}_i$. We need the following simple observation.

Observation 7.18 $\sum_{i=1}^q |E_i| \le \frac{\varphi \cdot \hat{W}}{\log^{30} N}$*, and* $|\hat{E}_3| \le \frac{\alpha_0 \cdot |T|}{2^{20}}$.

Proof: Fix an index $1 \le i \le q$. Since cut (A_i, B_i) is φ-sparse in G_{i-1}, $|E_i| \le \varphi \cdot \min\{w(A_i), w(B_i)\}$. Moreover, once cut (A_i, B_i) is processed, the total weight of all vertices in G decreases by at least $\min\{w(A_i), w(B_i)\} \ge \frac{|E_i|}{\varphi}$, and so at least $\frac{|E_i|}{\varphi}$ vertices are deleted from H'_i to obtain H_i. Since we have assumed that, by the time that cut (A_q, B_q) has been processed, fewer than $\frac{\hat{W}}{\log^{30} N}$ vertices were deleted from H, we get that $\sum_{i=1}^q |E_i| \le \frac{\varphi \cdot \hat{W}}{\log^{30} N}$.

From Observation 7.17, for all $1 \le i \le q$, $|\hat{\Gamma}_i| \le 2|E_i| \cdot \eta''$. Therefore:

$$|\hat{E}_3| = \sum_{i=1}^q |\hat{\Gamma}_i| \le 2\eta'' \cdot \sum_{i=1}^q |E_i| \le \frac{2\varphi \cdot \eta'' \cdot \hat{W}}{\log^{30} N} \le O\left(\frac{\hat{W}}{d' \cdot \log^{17} N}\right) \le \frac{\alpha_0 \cdot \hat{W}}{2^{21} \cdot d'} \le \frac{\alpha_0 \cdot |T|}{2^{20}},$$

since $\eta'' = \Theta(\log^{13} N)$, $\varphi = \frac{4}{d'}$, and N is large enough. $\square$

We are now ready to describe the type-3 updates to the expander X. Fix an index $1 \le i \le q$. In order to update expander X_{i-1} following the cut (A_i, B_i) in graph G_{i-1} that our algorithm produced, we start by deleting the edges of $\hat{\Gamma}_i$ from X_{i-1}. For each such edge $\hat{e} \in \hat{\Gamma}_i$, we consider every edge $e \in Q(\hat{e})$, and update the set $S'(e)$ of edges by deleting $\hat{e}$ from it. Next, we denote $T_i = V(X_{i-1}) \cap Z^*_i$, and $\overline{T}_i = V(X_{i-1}) \cap \overline{Z}^*_i$. If $\overline{T}_i \neq \emptyset$, then we delete the vertices of $\overline{T}_i$ from X_{i-1}, obtaining the final graph X_i. We now discuss this latter update in more detail, presenting a slightly different view of this update, that will be helpful for us later.

Assume first that (A_i, B_i) in G_{i-1} is a type-1 cut, so $Z_i = A_i \setminus J_i$. As observed already, in this case, graph $H'_i \setminus \Gamma_i$ contains no edge connecting a vertex of Z^*_i to a vertex of $\overline{Z}^*_i$. It is then easy to verify that graph $X_{i-1} \setminus \hat{\Gamma}_i$ contains no edge connecting a vertex of T_i to a vertex of $\overline{T}_i$. Indeed, if such an edge $\hat{e}$ existed, then its corresponding path $Q(\hat{e})$ would connect a vertex of Z^*_i to a vertex of $\overline{Z}^*_i$ in H'_i, and so it must contain an edge of Γ_i. But then $\hat{e} \in \hat{\Gamma}_i$ holds, a contradiction. Therefore, if (A_i, B_i) is a type-1 cut, and $T_i, \overline{T}_i \neq \emptyset$, then $(T_i, \overline{T}_i)$ is a cut of sparsity 0 in $X_{i-1} \setminus \hat{\Gamma}_i$.

Next, assume that (A_i, B_i) is a type-2 cut. As observed already, in this case, graph $H'_i \setminus \Gamma_i$ contains no edge connecting a vertex of $\overline{Z}^*_i$ to a vertex of Z^*_i. From the same arguments as above, graph $X_{i-1} \setminus \hat{\Gamma}_i$ contains no edge connecting a vertex of $\overline{T}_i$ to a vertex of $\overline{T}_i$. Therefore, if (A_i, B_i) is a type-2 cut, and $T_i, \overline{T}_i \neq \emptyset$, then $(\overline{T}_i, T_i)$ is a cut of sparsity 0 in $X_{i-1} \setminus \hat{\Gamma}_i$.

To conclude, we can view the update to graph X_{i-1} as follows: first, we delete all edges of $\hat{\Gamma}_i$ from X_{i-1}. Then we either delete all vertices from X_{i-1} (if $T_i = \emptyset$), or delete no vertices from X_{i-1} (if $\overline{T}_i = \emptyset$), or we obtain a sparsity-0 cut in the resulting graph $X_{i-1} \setminus \hat{\Gamma}$, and we delete all vertices on one side of this cut from X_{i-1}. This view of the update procedure will be helpful for us later. It is easy to verify that, at the end of this update, the collection $\{Q(\hat{e}) \in \mathcal{Q} \mid \hat{e} \in E(X_i) \setminus F\}$ defines a valid embedding of graph $X_i \setminus F$ into graph H_i. Indeed, we have ensured that $T_i \subseteq V(H_i)$. Additionally, for every edge $\hat{e} \in E(X_i) \setminus F$, both endpoints of $\hat{e}$ lie in set Z^*_i, and, since path $Q(\hat{e})$ is disjoint from the edges of Γ_i, it may not use vertices of $\overline{Z}^*_i$, and must therefore be contained in H_i.

Lastly, we would like to claim that $|V(X_q)|$ remains sufficiently large compared to $|T| = |V(X_0)|$, but unfortunately this may not be the case. Instead we will show that, if $1 \le i \le q$ is the smallest index, for which $|X_i| \le |T| \cdot \left(1 - \frac{8}{\log^4 N}\right)$ holds, then we can compute a finalizing cut in graph G_{i-1}. In our algorithm, we will monitor the number of vertices in the current graph X_i, and, if it ever falls below $|T|\cdot\left(1-\frac{8}{\log^4 N}\right)$, we will compute a finalizing cut, and apply the algorithm from Claim 7.8 to it, after which the current phase will be terminated, so in particular no further cuts (A_j,B_j) will be produced by the algorithm during the current phase. Therefore, essentially we only need to consider the case where $|V(X_{q-1})|\geq|T|\cdot\left(1-\frac{8}{\log^4 N}\right)$ but $|V(X_q)|<|T|\cdot\left(1-\frac{8}{\log^4 N}\right)$. The following claim summarizes the algorithm that will allow us to complete the current phase in such a case. For a vertex $v\in V(G_0)$, we denote by $\hat{w}(v)$ its weight at the beginning of current phase, and for a subset S of vertices of G_0, we denote by $\hat{w}(S)=\sum_{v\in S}\hat{w}(v)$.

Claim 7.19 *Suppose that $|V(X_{q-1})|\geq|T|\cdot\left(1-\frac{8}{\log^4 N}\right)$ but $|V(X_q)|<|T|\cdot\left(1-\frac{8}{\log^4 N}\right)$. Then there is a deterministic algorithm, that, given graphs G_0,X_0, and the sequence $(A_1,B_1),\ldots,(A_q,B_q)$ of cuts in G, and graph H_{q-1}, computes a finalizing cut $(\tilde{A},\tilde{B})$ in H_{q-1}. The running time of the algorithm is $O(\hat{W}^2)$.*

Proof: For convenience, in this proof we denote graph G_{q-1} by G' and graph H_{q-1} by H'. We compute the finalizing cut $(\tilde{A},\tilde{B})$ in H' as follows. First, for every vertex $v\in V(G')$, if $v\in A_q$, then we add v to $\tilde{A}$, and otherwise we add v to $\tilde{B}$. Next, we consider every special edge $e=(x,y)$ in G'. If $x,y\in A_q$, then we add all inner vertices on path $P^*(e)$ to $\tilde{A}$. If $x,y\in B_q$, then we add all inner vertices on path $P^*(e)$ to $\tilde{B}$.

Next, we consider two cases. The first case happens if (A_q,B_q) is a type-1 cut, so $V(G_q)=A_q\setminus J_q$. For each remaining special edge (that must have first endpoint in A_q and last endpoint in B_q), we add all inner vertices of $P^*(e)$ to $\tilde{B}$.

Note that, for every special edge e of G', the first and the last edges on path $P^*(e)$ are special edges, so the resulting cut $(\tilde{A},\tilde{B})$ in H' is weakly well-structured. Clearly:

$$|E_{H'}(\tilde{A},\tilde{B})|=|E_{G'}(A_q,B_q)|\leq\varphi\cdot\min\left\{w(A_q),w(B_q)\right\}\leq\frac{4}{d'}\cdot\frac{\hat{W}}{\log^{30}N}\leq\frac{8|T|}{\log^{30}N},$$

since $\varphi=\frac{4}{d'}$. We have also used the fact that, since the current phase has not terminated yet, $w(A_q),w(B_q)\leq\frac{\hat{W}}{\log^{30}N}$, and that, since Event $\mathcal{E}_1$ did not happen, $|T|\geq\frac{\hat{W}}{2d'}$.

Notice that, from our construction, $\tilde{A}=V(H_q)\cup J_q$. All vertices of $\tilde{B}$ are deleted from H' when the cut (A_q,B_q) is processed, and, since the current phase did not terminate yet, $|\tilde{B}|\leq\frac{\hat{W}}{\log^{30}N}$. Since $\tilde{A}=V(H_q)\cup J_q$, we get that $|\tilde{A}\cap T|\leq|T_q|+|J_q|$. Recall that we have assumed that $|T_q|=|V(X_q)|<|T|\cdot\left(1-\frac{8}{\log^4 N}\right)$. Additionally, $|J_q|\leq|E_{G'}(A_q,B_q)|\leq\frac{8|T|}{\log^{30}N}$. Altogether, we conclude that $|\tilde{A}\cap T|\leq|T|\cdot\left(1-\frac{7}{\log^4 N}\right)$, and $|\tilde{A}|\geq|V(H')|-\frac{\hat{W}}{32}$. Therefore, cut $(\tilde{A},\tilde{B})$ is a finalizing cut in graph H'.

The case where (A_q,B_q) is a type-2 cut is treated similarly, except that now, for every special edge e of G' whose endpoints belong to different sets in $\{A_q,B_q\}$, we place all inner vertices of $P^*(e)$ in $\tilde{A}$. Using the same reasoning as before, all edges of $E_{H'}(\tilde{A},\tilde{B})$ are special edges, and $|E_{H'}(\tilde{A},\tilde{B})|\leq\frac{8|T|}{\log^{30}N}$ holds. Our construction ensures that $\tilde{B}=V(H_q)\cup J'_q$. Using the same argument as above, we conclude that $|\tilde{B}\cap T|\leq|T_q|+|J'_q|\leq|T_q|+|E_{H'}(\tilde{A},\tilde{B})|\leq|T|\cdot\left(1-\frac{7}{\log^4 N}\right)$. Similarly, $|\tilde{B}|\geq|V(H')|-\frac{\hat{W}}{32}$, and so cut $(\tilde{A},\tilde{B})$ is finalizing. □

Whenever our algorithm computes a strongly well-structured φ-sparse cut (A_i,B_i) in the current graph G, we will first compute the corresponding sets E_i, Γ_i, and $\hat{\Gamma}_i$ of edges, together with the set T_i of terminals, before applying any changes to graphs G and H. If $|T_i| \geq |T| \cdot \left(1 - \frac{8}{\log^4 N}\right)$, then we proceed with updating the graphs G, H and X, as described above. But otherwise, we use the algorithm from Claim 7.19 in order to compute a finalizing cut $(\tilde{A}, \tilde{B})$ in the current graph H, and then employ the algorithm from Claim 7.8, that either correctly establishes that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ has happened (in which case we terminate the phase with a "FAIL"), or computes a strongly well-structured φ-sparse cut (A^*, B^*) in the current graph G, such that $w(A^*), w(B^*) \geq \frac{\hat{W}}{\log^{30} N}$. In the latter case, we do not process the cut (A_i, B_i), but instead output the cut (A^*, B^*), and update both graphs G and H accordingly, following which at least $\frac{\hat{W}}{\log^{30} N}$ vertices must be deleted from H, and the current phase terminates.

Combining All Three Types of Updates

We now complete our description of the algorithm for maintaining the expander X and the embedding of $X \setminus F$ into H throughout the phase. We will also maintain another graph X'. Initially, $X' = X$. Whenever a type-3 update (A_i, B_i) occurs, we update the graph X' as described above, but we will ignore type-1 and type-2 updates when maintaining X'. We will sometimes refer to X' as the "shadow expander", while we refer to X as the "expander", even though, as the algorithm progresses, it is possible that both graphs stop being expanders.

In order to maintain the graph X, whenever a type-1 or a type-2 update arrives, we process it exactly as describe above. Assume now that the ith type-3 update (A_i, B_i) arrives. In this case, we use graph X' in order to compute the set $\hat{\Gamma}_i$ of edges, and the partition $(T_i, \overline{T}_i)$ of the vertices of X', as described above, and we update graph X' accordingly. If, as the result, $|V(X')|$ falls below $|T| \cdot \left(1 - \frac{8}{\log^4 N}\right)$, then we invoke the algorithm from Claim 7.19, that computes a finalizing cut $(\tilde{A}, \tilde{B})$ in the current graph H. We then apply the algorithm from Claim 7.8 to the resulting cut. If the algorithm correctly estabishes that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ has happened, then we terminate our algorithm and return "FAIL". Otherwise, we obtain a strongly well-structured φ-sparse cut (A^*, B^*) in the current graph G, such that $w(A^*), w(B^*) \geq \frac{\hat{W}}{\log^{30} N}$. We output this cut, and update the graphs G and H accordingly, following which at least $\frac{\hat{W}}{\log^{30} N}$ vertices are deleted from H and the current phase terminates.

Assume now that $|V(X')| \geq |T| \cdot \left(1 - \frac{8}{\log^4 N}\right)$ continues to hold after the current update. We then update graph X as follows: we delete the edges of $\hat{\Gamma}_i$, and the vertices of $\overline{T}_i$ from it. This completes the description of the algorithm for maintaining graph X. It is easy to verify that, at all times, $X \subseteq X'$ holds. Therefore, when the ith type-3 update is processed, immediately after the edges of $\hat{\Gamma}_i$ are deleted from X, one of the cuts $(T_i \cap V(X), \overline{T}_i \cap V(X))$, $(\overline{T}_i \cap V(X), T_i \cap V(X))$ must be a cut of sparsity 0 in the resulting graph X.

Let $E^* = F \cup \hat{E}_1 \cup \hat{E}_2 \cup \hat{E}_3$. Recall that $|F| \leq \frac{\alpha_0 \cdot |T|}{500}$, and we have shown that $|\hat{E}_1| \leq \frac{\alpha_0 \cdot |T|}{5000}$ (in Claim 7.16), $|\hat{E}_2| \leq \frac{\alpha_0 \cdot |T|}{2^{20}}$, and $|\hat{E}_3| \leq \frac{\alpha_0 \cdot |T|}{2^{20}}$ (in Observation 7.18). Altogether, we get that $|E^*| \leq \frac{\alpha_0 \cdot |T|}{400}$ holds throughout the phase. We now prove that, throughout the phase, $|V(X)| \geq |T|/2$ holds.

Claim 7.20 *Throughout the phase,* $|V(X)| \geq |T|/2$.

Proof: Assume otherwise. Consider the first time τ at which $|V(X)| < |T|/2$ holds. For convenience, we denote by X_0 the original graph X, which is an α_0-expander, with $|T|$ vertices and we denote by $\tilde{X}$ the graph $X \setminus E^*$.

We construct the set $\mathcal{Y}$ of subsets of vertices of X_0, as follows. Initially, $\mathcal{Y} = \emptyset$. Whenever a type-2 update arrives, with a cut (A, B) of the current graph X, if $|A| \leq |B|$, then we add A to $\mathcal{Y}$ (recall that in this case, the vertices of A are deleted from X); otherwise, we add B to $\mathcal{Y}$. Clearly the cardinality of the set that we just added to $\mathcal{Y}$ is bounded by $|T|/2$. Moreover, since the edges of $E_X(A, B)$ belong to E^*, cut (A, B) has sparsity 0 in $\tilde{X}$.

Assume now that a type-3 update arrives, and let T' denote the set of vertices deleted from X in the current iteration. We then add T' to $\mathcal{Y}$. Since we did not terminate the current phase after the current iteration, $|V(X')| \geq |T| \cdot \left(1 - \frac{8}{\log^4 N}\right)$ must currently hold, and so $|T'| \leq \frac{8|T|}{\log^4 N} \leq \frac{|T|}{2}$ must hold. Moreover, as we have shown, one of the cuts $(T', V(X) \setminus T')$ or $(V(X) \setminus T', T')$ is a sparsity-0 cut in graph $\tilde{X}$.

Let $\mathcal{Y} = (Y_1, \ldots, Y_q)$ be the final collection of vertex subsets obtained after all updates that occured until time τ have been processed, where the sets are indexed in the order in wich they were added to $\mathcal{Y}$. We also add the set Y_{q+1} that contains all vertices that lie in X at time τ to $\mathcal{Y}$. From our assumption, $|X| \leq |T|/2$, and so for all $1 \leq q' \leq q+1$, $|Y_{q'}| \leq |T|/2$.

For all $1 \leq j \leq q$, let $Y'_j = Y_{j+1} \cup \cdots \cup Y_q$. At the time when Y_j was added to $\mathcal{Y}$, it defined a cut of sparsity 0 in graph $\tilde{X}$. In other words, either $E_{\tilde{X}}(Y_j, Y'_j) = \emptyset$, or $E_{\tilde{X}}(Y_j, Y'_j) = \emptyset$. Let α be such that $\max_{Y_j \in \mathcal{Y}} \{|Y_j|\} = \alpha \cdot |T|$. From our discussion so far, $\alpha \leq \frac{1}{2}$. From Observation 2.4, there is a cut (A, B) in $\tilde{X}$ with $|A|, |B| \geq |T|/4$, and $E_{\tilde{X}}(A, B) = \emptyset$.

Observe that $|E_{X_0}(A, B)| \leq |E^*| \leq \frac{\alpha_0 \cdot |T|}{400}$. Since $|A|, |B| \geq \frac{|T|}{4}$, we obtain a cut (A, B) in X_0, whose sparsity is below α_0, contradicting the fact that X_0 is an α_0-expander. □

7.5 Second Data Structure: Connecting to the Vertices of X

At the beginning of the phase, we initialize the algorithm for the ConnectToCenters problem from Theorem 6.7 on the current graph G, with $\beta = V(X)$, parameters d' and r remaining unchanged, and parameter N replaced by $N' = 8N$. We slightly modify the lengths of the special edges: for each such edge $e \in E^{\mathsf{spec}}$, we define the new length $\ell'(e) = \max\left\{\ell(e), 2\ell^0(e)\right\}$. We define, for every vertex $v \in V(G)$, a weight $w'(v)$ to be the length $\ell'(e)$ of the unique special edge e incident to v, and we let $W' = \sum_{v \in V(G)} w'(v)$. Clearly, $W' \leq 2\hat{W}$. As the lengths of special edges in the original graph G grow, we always set, for every edge $e \in E(G)$, the length $\ell'(e)$ that is used by the algorithm for the ConnectToCenters problem to $\ell'(e) = \max\left\{\ell(e), 2\ell^0(e)\right\}$, and we also define the weights $w'(v)$ of vertices $v \in V(G)$ with respect to these edge lengths. Clearly, for all $e \in E^{\mathsf{spec}}$, $\ell(e) \leq \ell'(e) \leq 2\ell(e)$ and for all $v \in V(G)$, $w(v) \leq w'(v) \leq 2w(v)$ always hold. We use W' to denote the sum of the initial weights $w'(v)$ for $v \in V(G)$, so it does not change over the course of the algorithm. Finally, the graphs G and the corresponding subdivided graph H' that serve as input to the ConnectToCenters problem must be given in the adjacency-list representation. We can compute both adjacency-list representations in time $O(|V(H')|^2) \leq O((W')^2) \leq O((W^0)^2)$. We prove that we obtain a valid input to the ConnectToCenters problem in the following simple claim, whose proof deferred to Section D.5 of Appendix.

Claim 7.21 *$(G, \{\ell'(e)\}_{e \in E(G)}, H', N', d', r, \beta)$ is a valid input to the* ConnectToCenters *problem.*

As the algorithm progresses, we will ensure that $\beta = V(X)$ always holds. Since, from Claim 7.20, $|V(X)| \geq |T|/2$ holds throghout the phase, this will ensure that at most half the vertices may be deleted from the initial set β over the course of the phase. From Observation 7.2, $\sum_{e \in E^{\mathsf{spec}}} \ell^*(e) \leq 2\sum_{e \in E^{\mathsf{spec}}} \ell^0(e) + 2n$. Therefore, the total increase in the lengths $\ell'(e)$ of edges $e \in E(G)$ over the course of the entire algorithm is at most $2n \leq 8|V(G)|$, as required.

Consider the graph H', that is the subdivided graph corresponding to G and the lengths $\ell'(e)$ of its edges, and recall that H is the subdivided graph corresponding to G and the lengths $\ell(e)$ of its edges. Recall that, for every edge $e \in E(G)$, $\ell(e) \leq \ell'(e) \leq 2\ell(e)$. Since the lengths of all special edges are integral powers of 2, for every special edge $e \in E(G)$, either $\ell'(e) = \ell(e)$ or $\ell'(e) = 2\ell(e)$ holds. Therefore, graph H' can be obtained from graph H as follows: for every edge $e \in E^{\mathsf{spec}}$ with $\ell'(e) = 2\ell(e)$, we subdivide every special edge $\hat{e} \in E(H)$ that lies on the path $P^*(e)$ representing e once, obtaining a path $\hat{P}(\hat{e})$ with 2 special edges, representing edge $\hat{e}$ in H'. For all edges $\hat{e} \in E(H') \cap E(H)$, we let $\hat{P}(\hat{e}) = (\hat{e})$.

Recall that the algorithm for the $\mathsf{ConnectToCenters}$ problem may, at any time, produce Outcome O3, namely, a weakly well-structured cut (A', B') in graph H', with $|A'| \geq \frac{511|V(H')|}{512}$, $|A' \cap V(X)| < \frac{2|V(X)|}{\log^3 N'}$, and $|E_{H'}(A', B')| \leq \frac{|V(H')|}{1024d' \cdot \log^6 N'}$. Consider the cut (A, B) in H with $A = A' \cap V(H)$ and $B = B' \cap V(H)$. We claim that (A, B) is a finalizing cut in H, in the following simple observation.

Observation 7.22 *Cut (A, B) is a finalizing cut in H.*

Proof: Since, at the beginning of the phase, $|V(H')| = W' \leq 2\hat{W}$, and over the course of the phase, the total increase in the lengths $\ell'(e)$ of edges $e \in E(G)$, and hence in $|V(H')|$, is at most $4n \leq 8\hat{W}$, we get that $|V(H')| \leq 10\hat{W}$ holds throughout the phase.

Consider any edge $e = (x, y) \in E_H(A, B)$. First, we claim that e is a special edge. Indeed, otherwise, $e \in E_{H'}(A', B')$ must hold, contradicting the fact that (A', B') is a weakly well-structured cut. If $e \in E(H')$, then $e \in E_{H'}(A', B')$ must hold. Otherwise, at least one of the two special edges that lie on the path $\hat{P}(e)$ representing e in H' belong to $E_H(A, B)$. Therefore:

$$|E_H(A, B)| \leq |E_{H'}(A', B')| \leq \frac{|V(H')|}{1024d' \cdot \log^6 N'} \leq \frac{10\hat{W}}{1024d' \cdot \log^6 N} \leq \frac{|T|}{8 \log^6 N},$$

since $|T| \geq \frac{\hat{W}}{2d'}$, as Event $\mathcal{E}_1$ did not happen. Recall that $|A'| \geq \frac{511|V(H')|}{512}$, and so:

$$|B| \leq |B'| \leq \frac{|V(H')|}{512} \leq \frac{10\hat{W}}{512} \leq \frac{\hat{W}}{32}.$$

Therefore, $|A| \geq |V(H)| - \frac{\hat{W}}{32}$ must hold. Lastly, $|A \cap V(X)| \leq |A' \cap V(X)| < \frac{2|V(X)|}{\log^3 N'} < \frac{|T|}{2}$. We conclude that (A, B) is a finalizing cut in H. □

For brevity, whenever the algorithm for the $\mathsf{ConnectToCenters}$ on G produces Outcome O3, we say that it produces a finalizing cut (A, B) in H.

Recall that the algorithm for the $\mathsf{ConnectToCenters}$ problem uses a parameter $\rho = \frac{2^{12} \cdot W' \cdot \log^3 N'}{|T|} \leq O\left(\frac{\hat{W} \log^3 N}{|T|}\right)$, and that the total increase in the lengths of all edges is denoted by Λ; in our case, $\Lambda \leq O(n)$ holds. Recall also that β' denotes the set $V(X)$ of vertices at the end of the algorithm. The running time of the algorithm from Theorem 6.7 for this instance of the $\mathsf{ConnectToCenters}$ problem is then bounded by:

$$
\begin{aligned}
O\left(c_1 \cdot \hat{W} \cdot (\hat{W} + |T| \cdot \rho + \Lambda) \cdot 2^{c_2\sqrt{\log N'}} \cdot (\log N')^{16c_2(r-1)+8c_2+26}\right) \\
\leq O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N'}} \cdot (\log N')^{16c_2(r-1)+8c_2+29}\right) \\
\leq O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+30} \cdot \left(1 + \frac{1}{32c_2 r}\right)^{16c_2(r-1)+8c_2+30}\right) \\
\leq O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+30}\right).
\end{aligned}
$$

(we have used Inequalities 17 and 18.)

We denote the data structure for the ConnectToCenters problem that we described above by DS^{out}, and we denote the set $\mathcal{Q} = \bigcup_{i=1}^{\lambda-1} \mathcal{Q}_i$ of paths that it maintains by $\mathcal{Q}^{\text{out}}$. If a vertex v lies in layer U_λ, then we say that v is *disconnected* in DS^{out}. For each vertex $v \in \bigcup_{i=1}^{\lambda-1} U_i$, the unique path of $\mathcal{Q}_i$ that originates at v is denoted by $K(v)$.

Next, we consider the graph $\overleftarrow{G}$, obtained from G by reversing the direction of its edges, and its corresponding subidivided graph $\overleftarrow{H}$. It is easy to verify that $\overleftarrow{G}$ is a perfect well-structured graph. We define an instance of the ConnectToCenters problem on graph $\overleftarrow{G}$, with the edge lengths $\{\ell'(e)\}_{e \in E(\overleftarrow{G})}$, set β of vertices, and parameters N', d' and r defined exactly as before, and then apply the algorithm for the ConnectToCenters problem from Theorem 6.7 to this instance. We denote the data structure maintained by this algorithm by DS^{in}. If a vertex v lies in layer U_λ, then we say that v is *disconnected* in DS^{in}. In order to avoid confusion, we denote the layers of this data structure by $U'_0, \ldots, U'_\lambda$. Recall that, for all $1 \leq i < \lambda$, the algorithm maintains a collection $\mathcal{Q}_i = \{K(v) \mid v \in U'_i\}$ of paths in graph $\overleftarrow{H}$, where for all $v \in U_i$, path $K(v)$ connects v to some vertex of $\bigcup_{i'<i} U'_{i'}$. By reversing the direction of edges in this path, we obtain a path $K'(v)$ in H, that connects some vertex of $\bigcup_{i'<i} U'_{i'}$ to v. We denote by $\mathcal{Q}'_i = \{K'(v) \mid v \in U'_i\}$, and by $\mathcal{Q}^{\text{in}} = \bigcup_{i=1}^{\lambda-1} \mathcal{Q}'_i$. To avoid confusion, for a vertex $v \notin U_\lambda$, we use $K(v)$ only to denote the path maintained by DS^{out}, that is, $K(v)$ is the unique path originating from v in $\mathcal{Q}^{\text{out}}$, and for $v \notin U'_\lambda$, we use $K'(v)$ to denote the unique path terminating at v in $\mathcal{Q}^{\text{in}}$.

If the algorithm for the ConnectToCenters problem in graph $\overleftarrow{G}$ produces Outcome O3, then, using the same reasoning as in the proof of Observation 7.22, we obtain a finalizing cut (A, B) in graph $\overleftarrow{H}$. It is then immediate to verify that (B, A) is a finalizing cut in graph H. For brevity, whenever the algorithm for the ConnectToCenters problem in graph $\overleftarrow{G}$ produces Outcome O3, we say that it produces a finalizing cut in H.

Assume now that the algorithm for the ConnectToCenters problem in graph $\overleftarrow{G}$ produces Outcome O2; that is, it computes a strongly well-structured $\frac{1}{d'}$-sparse cut (A, B) in $\overleftarrow{G}$. It is then easy to see that (B, A) is a strongly well-structured $\frac{1}{d'}$-sparse cut in G. Therefore, if the algorithm for the ConnectToCenters problem in graph $\overleftarrow{G}$ produces Outcome O2, we will say that it produces a strongly well-structured $\frac{1}{d'}$-sparse cut in G.

From our discussion, the total running time of the algorithms that maintain data structures DS^{in} and DS^{out}, including the time that is needed in order to construct the corresponding two instances of the ConnectToCenters problem, and including the time that is needed in order to compute a finalizing cut in H, if any of the two algorithms produce Outcome O3, is bounded by:

$$O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+30}\right).$$

7.6 Additional Data Structures

Recall that our algorithm maintains a graph X (that is initially an expander), a subset F of edges of X (the fake edges), and an embedding $\mathcal{Q}$ of $X \setminus F$ into H. Additionally, it maintains a *shadow expander* X', which is initially identical to X, but, as the time progresses, we may ignore some of the updates to graph X and only apply a subset of such updates to X'. We also maintain a subset F' of edges of X', which is initially identical to F, and an embedding $\mathcal{Q}'$ of $X' \setminus F'$ into H, which is initially identical to $\mathcal{Q}$. Throughout the algorithm, $X' \subseteq X$ and $F' \subseteq F$ hold. Additionally, for every edge $e \in E(X') \setminus F$, its embedding path $Q'(e) \in \mathcal{Q}'$ is identical to the embedding path $Q(e) \in \mathcal{Q}$. As the time progresses, vertices and edges may be deleted from X and from X'. Whenever an edge $e \in F$ is deleted from X, we delete it from F as well, and similarly, whenever an edge $e' \in F'$ is deleted from X', we delete it from F' as well. For every edge $e \in E(X)$, its embedding path $Q(e) \in \mathcal{Q}$ remains unchanged throughout the algorithm, except that, whenever some special edge on path $Q(e)$ undergoes splitting, we modify the path accordingly. Embedding $\mathcal{Q}'$ of $X' \setminus F'$ into H is maintained similarly.

For every edge $e \in E(X) \setminus F$, we maintain the length $\hat{\ell}(e)$ of the embedding path $Q(e)$. We initialize the value $\hat{\ell}(e)$ at the beginning of the phase, once the graph X and the embedding $\mathcal{Q}$ of $X \setminus F$ into H are computed. After that, whenever a special edge of $Q(e)$ undergoes splitting, we increase $\hat{\ell}(e)$ by (additive) 2. The time that is required in order to maintain the values $\hat{\ell}(e)$ for edges $e \in E(X) \setminus F$ is clearly subsumed by the time required to compute and to maintain the collection $\mathcal{Q}$ of paths. In our description of the remainder of the algorithm below, we will ignore the task of maintaining these values.

For every edge $e \in E(H)$, we also maintain a list $S(e)$ of all edges $\hat{e} \in E(X) \setminus F$, such that $e \in Q(\hat{e})$. The task of maintaining these lists is quite straightforward; we will ignore it when we describe the remainder of the algorithm below, but we will show how to extend the algorithm in order to maintain these lists efficiently later.

Throughout, we use a distance parameter $\hat{d} = \frac{2^{30} \log \hat{W}}{\alpha_0}$. Throughout the phase, we maintain an arbitrary vertex $v \in V(X)$, and two shortest-paths trees $\hat{\tau}'$ and $\hat{\tau}''$ rooted at v in graph $X \setminus F$, where the edges in $\hat{\tau}'$ are directed away from v, and the edges in $\hat{\tau}''$ are directed towards v. In other words, for every vertex $a \in V(X)$, the unique s-a path in $\hat{\tau}'$ is a shortest s-a path in $X \setminus F$, and the unique a-s path in $\hat{\tau}''$ is a shortest a-s path in $X \setminus F$. Note that both trees can be computed in time $O(|T|)$, since $|V(X)| \leq |T|$, and maximum vertex degree in X is $O(1)$. Whenever an edge or a vertex are deleted from X, we recompute both trees from scratch, selecting a new root vertex $v \in V(X)$ if needed. Since $|E(X)| \leq O(|T|)$, the trees need to be recomputed at most $O(|T|)$ times, and so the total time spent over the course of the phase on computing the two trees is bounded by $O(|T|^2) \leq O((W^0)^2)$.

7.7 The Remainder of the Algorithm

In this subsection we describe the remainder of the algorithm for a single phase of the algorithm for the MaintainCluster problem. At the beginning of the phase, we construct the subdivided graph H corresponding to the initial graph G, and then construct the set T of terminals, as described in Section 7.2. We check whether either of the bad events $\mathcal{E}_1$ or $\mathcal{E}_4$ happened, and if so, we terminate the phase and return "FAIL". We assume from now on that neither of these events happened. The running time of the algorithm so far is $O(|E(H)|) \leq O((W^0)^2)$.

We execute the algorithm from Lemma 7.11 in order to construct the expander X. If the algorithm terminates with a "FAIL", which may only happen with probability at most $\frac{1}{N^7}$, then we terminate the algorithm for the current phase and return "FAIL". If the algorithm returns a finalizing cut in H, then we apply the algorithm from Claim 7.8 to this cut. If the algorithm from Claim 7.8 estabishes that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ happened, we terminate the phase with a "FAIL". Otherwise, we obtain a strongly well-structured φ-sparse cut (A^*, B^*) in the current graph G, such that $w(A^*), w(B^*) \geq \frac{\hat{W}}{\log^{30} N}$. In this case, we output the cut (A^*, B^*). Once this cut is processed, at least $\frac{\hat{W}}{\log^{30} N}$ vertices are deleted from H, and the current phase terminates. Finally, if the algorithm from Lemma 7.11 returns a strongly well-structured $1/d'$-sparse cut (A, B) in G with $w(A), w(B) \geq \frac{\hat{W}}{(\log N)^{30}}$, then as before we output this cut, and, once it is processed, at least $\frac{\hat{W}}{\log^{30} N}$ vertices are deleted from H, and the current phase terminates. Therefore, we assume from now on that the algorithm from Lemma 7.11 computed a graph X with $V(X) = T$, whose maximum vertex degree is at most 24, such that X is an $\frac{\alpha_0}{81}$-expander. Recall that we are also given a set $F \subseteq E(X)$ of at most $\frac{\alpha_0 \cdot |T|}{500}$ fake edges, and an embedding $\mathcal{Q}$ of $X \setminus F$ into H. Every path in $\mathcal{P}^*$ has length at most d'', and the congestion caused by the paths in $\mathcal{P}^*$ is at most η''. The running time of the algorithm from Lemma 7.11 is bounded by:

$$O\left(c_1 \cdot \hat{W}^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+3}\right) \leq O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+3}\right),$$

and the running time of the algorithm from Claim 7.8 is $O(\hat{W}^2) \leq O((W^0)^2)$.

We also initialize data structures DS^{in} and DS^{out}. Recall that the total time for maintaining both data structures is bounded by:

$$O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+30}\right).$$

Recall that the algorithm for the $\mathsf{ConnectToCenters}$ problem, that is used to maintain data structures DS^{in} and DS^{out} may return "FAIL" with probability at most $\frac{1}{N^7}$. If any of these two instantiations of the algorithm terminate with a "FAIL", we terminate the algorithm for the current phase, and return "FAIL".

The running time of the algorithm so far, including the initialization of all data structures, and maintaining data structures DS^{in} and DS^{out} throughout the phase, is bounded by:

$$O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+30}\right).$$

Our algorithm employs a cleanup procedure, that we describe below, once at the beginning of the phase, and once after each query is processed. The purpose of the procedure is to ensure that the following conditions hold:

C1. The length of every path in $\mathcal{Q}$ is bounded by $8d''$;

C2. The depth of each of the trees $\hat{\tau}', \hat{\tau}''$ is at most $\hat{d}$; and

C3. $U_\lambda = \emptyset$ and $U'_\lambda = \emptyset$.

(here, U_λ and U'_λ are the sets of disconnected vertices from data structures DS^{out} and DS^{in}, respectively). Before we describe the cleanup procedure that ensures all these conditions, we describe our algorithm for responding to queries.

7.7.1 Responding to Queries

We assume that, at the time when a query (x, y) arrives, Conditions C1–C3 hold. We start by computing a path P_1 in graph H, connecting x to some vertex $x' \in V(X)$, as follows. Since $U_\lambda = \emptyset$ currently holds, there is some layer $0 \le i < \lambda$ in data structure DS^{out}, with $x \in U_i$. If $i = 0$, then we let the path P_1 contain a single vertex – vertex x itself. Otherwise, we use the unique path $K(x) \in \mathcal{Q}_i$ to connect x to some vertex $x_1 \in U^{<i}$. Let U_{i_1} be the layer in which x_1 lies, so $i_1 < i$. We say that $K(x)$ is the first *subpath* of P_1. If $x_1 \in V(X)$, then we have completed the construction of the path P_1. Otherwise, using a similar process, we connect x_1 to some vertex $x_2 \in U^{<i_2}$, via the path $K(x_1)$, that becomes the second subpath of P_1 and so on. After at most $\lambda - 1$ iterations, we are then guaranteed to reach a vertex $x' \in V(X)$, thereby obtaining a path P_1 connecting x to x'. Since the length of every path in $\mathcal{Q}^{\text{out}}$ is at most $16d' \log^{10} N'$ (from the relaxed condition L1), while $d' \le \frac{2d}{(\log N)^{28}}$ and $\lambda \le O(\log N)$, we get that the length of P_1 is bounded by:

$$16d'\lambda \log^{10} N' \le O\left(\frac{d}{(\log N)^{17}}\right).$$

Using a similar algorithm, together with data structure DS^{in}, we compute a path P_2 in graph H, of length at most $O\left(\frac{d}{(\log N)^{17}}\right)$, connecting some vertex $y' \in V(X)$ to y. The time required to compute the paths P_1 and P_2 is bounded by $O(|E(P_1)| + |E(P_2)|)$.

Next, we use the tree $\hat{\tau}''$ to compute a path P_3' in $X \setminus F$, connecting x' to the root v of the tree $\hat{\tau}''$, and we use the tree $\hat{\tau}'$ to compute a path P_4' in $X \setminus F$ connecting v to y'. By concatenating both paths, we obtain a path P' in $X \setminus F$ that connects x' to y', whose length is at most $2\hat{d}$.

Denote $P' = (e_1, \ldots, e_z)$. We compute a path P_3 connecting x' to y' in graph H by concatenating the corresponding embedding paths $Q(e_1), \ldots, Q(e_z)$ in $\mathcal{Q}$. Recall that, from Condition C1, the length of each such path is bounded by $8d''$. Since $\hat{d} = \frac{2^{30} \log \hat{W}}{\alpha_0} \le O(\log N)$ and $d'' = \Theta\left(d' \cdot (\log N)^{16}\right) \le O\left(\frac{d}{(\log N)^{12}}\right)$, as $d' \le \frac{2d}{(\log N)^{28}}$, we get that the length of the resulting path P_3 is bounded by:

$$2\hat{d} \cdot 8d'' \le O\left(\frac{d}{(\log N)^{11}}\right).$$

The time required to compute the path P_3 is bounded by $|E(P_3)|$. By concatenating the paths P_1, P_3 and P_2, we obtain a path P' connecting x to y in H, whose length is bounded by:

$$O\left(\frac{d}{(\log N)^{17}}\right) + O\left(\frac{d}{(\log N)^{11}}\right) < d,$$

since N is sufficiently large. The total time spent on computing path P' is $O(|E(P')|) \le O(d)$. We then turn P' into a simple x-y path of length at most d in graph H, in time $O(d)$. Finally, we convert P' into a simple x-y path in graph G of length at most d in a straightforward way, in time $O(d)$, and return the path P.

The time to respond to a single query is bounded by $O(d) \le O(n) \le O(W^0)$, and the time to respond to all queries in a single phase is bounded by $O(W^0 \cdot \Delta') \le O\left(\frac{W^0 \cdot \Delta}{(\log N)^{30}}\right)$ (we have used the fact that $\Delta' = \frac{\Delta}{\log^{30} N}$, and Inequality 15).

7.7.2 Post-Query Updates

Recall that, after responding to a query, the algorithm receives a subset $E' \subseteq E(P)$ of special edges on the path P that was returned in response to the query, whose lengths need to be doubled. We can then compute a subset $E'' \subseteq E'$ of edges, whose lengths $\ell'(e)$ that are used by the algorithms that maintain data structures DS^{in} and DS^{out} need to be doubled, and supply the set E'' of edges to these two algorithms, in time $O(|E'|)$. Next, we consider the edges in E' one by one. For each edge $e \in E'$, we consider every special edge e' on the path $P^*(e)$ representing e in H. We split each such edge $e' = (x, y)$, by replacing it with the path (x, x', y', y) in H. If any of the endpoints x, y of e' are terminal vertices that lie in T, this operation does not affect them. Notice that this operation inserts two new special edges into graph H: edge $e'_1 = (x, x')$ and edge $e'_2 = (y, y')$. We initialize the lists $S(e'_1) = S(e'_2) = \emptyset$.

Next, we consider every edge $\hat{e} \in S(e')$, one by one. For each such edge $\hat{e}$, we update the embedding path $Q(\hat{e})$ by splitting the edge e', and also update the length $\hat{\ell}(e)$ of the path by increasing it by (additive) 2. We also insert $\hat{e}$ into $S(e'_1)$ and $S(e'_2)$. If the length $\hat{\ell}(e)$ of e became greater than $8d''$, then we flag this edge, to be taken care of by the Cleanup procedure.

In order to bound the time required for post-query updates, let E^* be the set of all special edges that ever belonged to graph H over the course of the phase. Since, from Observation 7.2, $\sum_{e \in E^{\mathsf{spec}}} \ell^*(e) \leq 2\sum_{e \in E^{\mathsf{spec}}} \ell^0(e) + 4n \leq O(W^0)$, and since, for each edge $e \in E(G)$, the number of its child edges that ever belonged to H over the course of the phase is asymptotically bounded $\ell^*(e)$, we get that $|E^*| \leq O(W^0)$.

Recall that the congestion caused by the embedding $\mathcal{Q}$ of $X \setminus F$ into H is bounded by $O((\log N)^{13})$. Therefore, for every edge in E^* we spend $O((\log N)^{13})$ time to process it when it is first inserted into H. The total time required for all post-query updates is then bounded by:

$$O(W^0 \cdot (\log N)^{13}).$$

The total running time of the algorithm so far, including responding to queries, but excluding the Cleanup Procedure, is bounded by:

$$O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+30}\right) + O\left(\frac{W^0 \cdot \Delta}{(\log N)^{30}}\right).$$

7.7.3 The Cleanup Procedure

The cleanup procedure is executed once at the beginning of the phase, and then after every query, once all edge-splittings that follow the query are processed. The purpose of the procedure is to ensure that Properties C1–C3 are satisfied. As part of the procedure, we will sometimes compute φ-sparse strongly well-structured cuts in graph G. Whenever this happens, we will call to another procedure, called $\mathsf{ProcessCut}'$, that will process the resulting cut. We view Procedure $\mathsf{ProcessCut}'$ as part of the cleanup procedure, and we describe it in the following subsection. We note that, if the cleanup procedure ever identifies a φ-sparse strongly well-structured cut $(A.B)$ in G, our algorithm does not immediately output the cut, but rather calls to Procedure $\mathsf{ProcessCut}'$. The procedure may choose to either output the cut (A, B), after which the vertices of the side of the cut with smaller weight and the endpoints of the edges in $E_G(A, B)$ are deleted from G, and graphs H, X and X' are updated accordingly; or it may instead compute another finalizing cut (A', B') in H, and then call to Claim 7.8 to process it. In the latter case the algorithm does not output the initial cut (A, B), but after cut

(A', B') is processed the phase will terminate. We now turn to describe the cleanup procedure.

The procedure consists of iterations, which are executed as long as at least one of the conditions C1–C3 is violated.

Assume first that Condition C3 is violated. In this case we say that the current iteration is of type 1. Assume w.l.o.g. that $U_\lambda \neq \emptyset$; the other case is symmetric. We execute a single iteration of the algorithm for the $\mathsf{ConnectToCenters}$ problem that maintains data structure DS^{out}. We now consider three cases. If the algorithm produced Outcome O1, that is, it obtained a valid $\mathsf{Layered\ Connectivity\ Data\ Structure}$ with $U_\lambda = \emptyset$, then the current iteration terminates. If the algorithm produced a finalizing cut (A', B') in the current graph H, then the cut processed like decribed in Section 7.5, after which the current phase terminates. Recall that the running time for processing the cut (A', B') is included in our analysis of the running time of the algorithm that maintains data structure DS^{out}. Lastly, if the algorithm for the $\mathsf{ConnectToCenters}$ problem produces a a strongly well-structured $\frac{1}{d'}$-sparse cut (A, B) in G, then we call to Procedure $\mathsf{ProcessCut}'$. The procedure will either produce a finalizing cut in H, that will subsequently terminate the current phase; or it will output the cut (A, B), following which, if $w(A) \leq w(B)$, then the vertices of A, and the endpoints of the edges in $E_G(A, B)$ are deleted from G, while otherwise, the vertices of B, and the endpoints of the edges in $E_G(A, B)$ are deleted from G, and the current iteration terminates. The total running time of all type-1 iterations of the Cleanup procedure executed over the course of the phase, excluding the running time of Procedure $\mathsf{ProcessCut}'$, is already included in the running times of the algorithms that maintain data structures DS^{in} and DS^{out}.

Assume now that Condition C1 is violated. In this case we say that the current iteration is of type 2. Let $\hat{e} \in E(X)$ be the edge with $\hat{\ell}(\hat{e}) > 8d''$; recall that all such edges were flagged by the algorithm that performed post-query updates. We then simply delete the edge $\hat{e}$ from graph X. We also recompute the shortest path threes $\hat{\tau}'$ and $\hat{\tau}''$ in $X \setminus F$ that are rooted at the same vertex v, in time $O(|E(X)|) \leq O(|T|) \leq O(W^0)$. The total running time of type-2 iterations over the course of the whole phase, excluding the time needed for recomputing the trees $\hat{\tau}'$ and $\hat{\tau}''$, is asymptotically bounded by the running time of the algorithm that performs post-query updates after every query. Since at least one edge is deleted from X in each type-2 iteration, and $|E(X)| \leq O(|T|) \leq O(W^0)$, the total time spent on recomputing the trees $\hat{\tau}'$ and $\hat{\tau}''$ in all type-2 iterations of the phase is bounded by $O((W^0)^2)$.

Lastly, assume that Condition C2 is violated. In this case we say that the current iteration is of type 3. Then we can compute, in time $O(1)$, two vertices a pair $a, b \in V(X)$ with $\mathsf{dist}_{X \setminus F}(a, b) > \hat{d}$. Recall that $\hat{d} = \frac{2^{30} \log \hat{W}}{\alpha_0}$, and the maximum vertex degree in X is bounded by $D = 24$. We can now use the algorithm from Observation 2.3 to compute a cut (A, B) in $X \setminus F$ of sparsity at most $\varphi' = \frac{32 D \log \hat{W}}{\hat{d}} \leq \frac{\alpha_0}{2^{20}}$. Let $Z \in \{A, B\}$ be the set of vertices of smaller cardinality, breaking ties arbitrarily. For every vertex $x \in Z$, we delete x from $V(X)$. We supply the list of all vertices that are deleted from X to the algorithms for the $\mathsf{ConnectToCenters}$ problem that are used to maintain data structures DS^{in} and DS^{out}. Recall that the running time of the algorithm from Observation 2.3 is bounded by $O(D \cdot |V(X)|) \leq O(\hat{W}) \leq O(W^0)$. We also select an arbitrary vertex $v \in V(X)$, and compute shortest-path trees $\hat{\tau}'$ and $\hat{\tau}''$ in $X \setminus F$ rooted at v. Note that both trees can be computed in time $O(|E(X)|) \leq O(\hat{W}) \leq O(W^0)$. Every time a type-3 iteration occurs, at least one vertex is deleted from X. Therefore, the total time that the algorithm spends on type-3 iterations over the course of the entire phase is bounded by $O((W^0)^2)$.

This completes the algorithm for the Cleanup procedure. From our discussion, the total running time of the algorithm for the $\mathsf{MaintainCluster}$ problem so far, excluding Procedure $\mathsf{ProcessCut}'$, is bounded by:

$$O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+30}\right) + O\left(\frac{W^0 \cdot \Delta}{(\log N)^{30}}\right).$$

Next, we describe the algorithm for Procedure $\mathsf{ProcessCut}'$.

7.7.4 Procedure $\mathsf{ProcessCut}'$

The input to Procedure $\mathsf{ProcessCut}'$ is a strongly well-structured φ-sparse cut (A, B) in graph G. Recall that, if our algorithm chooses to output the cut (A, B), then the vertices of the side of the cut that has smaller weight will be deleted from G, together with the endpoints of the edges in $E_G(A, B)$. After that, graph H will be updated with a corresponding sequence of vertex- and edge-deletions, and expander graph X, together with the shadow expander X' will also be updated, as described in Section 7.4.2. If we apply all these updates directly, we may discover that too many vertices have been deleted from graph X'. In such a case, we could use the algorithm from Claim 7.19, in order to compute a finalizing cut $(\tilde{A}, \tilde{B})$ in graph H. However, $(\tilde{A}, \tilde{B})$ is a valid finalizing cut in the original graph H, to which the updates following the cut (A, B) have not been applied yet. Therefore, our algorithm will, instead, simulate the updates to graphs G, H and X' in order to find out whether the number of vertices that are deleted from X' following these updates is too high. If so, it will apply the algorithm from Claim 7.19 to compute a finalizing cut $(\tilde{A}, \tilde{B})$ in graph H, followed by the algorithm from Claim 7.8, that either correctly establishes that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ has happened (in which the current phase terminates with "FAIL"), or computes a strongly well-structured φ-sparse cut (A^*, B^*) in the current graph G, such that $w(A^*), w(B^*) \geq \frac{\hat{W}}{\log^{30} N}$, which our algorithm then outputs; in this case, our algorithm does not output the cut (A, B). The vertices of the side of the cut (A^*, B^*) that has smaller weight are then deleted from G, together with the endpoints of the edges of $E_G(A^*, B^*)$, and the current phase terminates. If our algorithm establishes that $|X'|$ remains sufficiently high even after we update the graphs G, H and X' with the cut (A, B), then it will simply output the cut (A, B), and update the graphs G, H, X' and X accordingly.

We now provide a formal description of Procedure $\mathsf{ProcessCut}'$. We assume that our algorithm maintains the graph G_0, which is the graph G at the beginning of the current phase, and the sequence of strongly well-structured φ-sparse cuts $(A_1, B_1), \ldots, (A_q, B_q)$ that it has output so far over the course of the phase. We also assume that it maintains the invariant that $|V(X')| \geq |T| \cdot \left(1 - \frac{8}{\log^4 N}\right)$. This invariant clearly holds at the beginning of the phase, when $V(X') = T$.

We now assume that Procedure $\mathsf{ProcessCut}'$ is called with a strongly well-structured φ-sparse cut (A, B) in the current graph G. We assume that so far the algorithm returned $q \geq 0$ well-structured φ-sparse cuts, that we denote by $(A_1, B_1), \ldots, (A_q, B_q)$, in the order in which the algorithm returned them. We also denote the cut (A, B) by (A_{q+1}, B_{q+1}), and let $E_{q+1} = E_G(A_{q+1}, B_{q+1})$. Let $J_{q+1} \subseteq A_{q+1}, J'_{q+1} \subseteq B_{q+1}$ the vertices that serve as endpoints of the edges in E_{q+1}. As before, if $w(A_{q+1}) \geq w(B_{q+1})$, then we denote $Z_{q+1} = A_{q+1} \setminus J_{q+1}$, and otherwise we denote $Z_{q+1} = B_{q+1} \setminus J'_{q+1}$. We also let $\overline{Z}_{q+1} = V(G) \setminus Z_{q+1}$ contain the remaining vertices of the current graph G. As before, if $Z_{q+1} = A_{q+1} \setminus J_{q+1}$, then we say that cut (A_{q+1}, B_{q+1}) is of *type 1*, and otherwise we say that it is of *type 2*. We also denote by G_q and H_q the graphs G and H, respectively, at the beginning of Procedure $\mathsf{ProcessCut}'$, and we denote $z_{q+1} = \min\{w(A), w(B)\}$.

For every edge $e = (x, y) \in E_{q+1}$, we compute the set $\Gamma(e)$ of edges in H_q. Recall that the set $\Gamma(e)$ consists of all vertices on the path $P^*(e)$ that represents e in graph H_q, as well as all regular edges insident to vertices x and y in H_q. It is easy to verify that $|\Gamma(e)| \leq |V(H_q)| \leq O(\hat{W})$. Next, we compute the set $\Gamma_{q+1} = \bigcup_{e \in E_{q+1}} \Gamma(e)$ of edges in $\hat{H}$. The sets $\Gamma(e)$ of edges for all $e \in E_{q+1}$, together with the set Γ_{q+1} of edges can be computed in time $O(|E_{q+1}| \cdot \hat{W}) \leq O(\varphi z_{q+1} \cdot \hat{W})$.

We also compute the partition of the vertices of H_q into sets Z^*_{q+1} and $\overline{Z}^*_{q+1}$, as follows. Set $\overline{Z}^*_{q+1}$ contains all vertices that lie in set $\overline{Z}_{q+1}$. Additionally, for every special edge e of G with both endpoints in $\overline{Z}_{q+1}$, set $\overline{Z}^*_{q+1}$ contains all vertices on path $P^*(e)$. Similarly, set Z^*_{q+1} contains all vertices that lie in set Z_{q+1}, and additionally, for every special edge e of G with both endpoints in Z_{q+1}, set Z^*_{q+1} contains all vertices on path $P^*(e)$. It is easy to verify that $(Z^*_{q+1}, \overline{Z}^*_{q+1})$ is a partition of $V(H_{q+1})$, since cut (A_{q+1}, B_{q+1}) is strongly well-structured in G. The time required to compute the partition $(Z^*_{q+1}, \overline{Z}^*_{q+1})$ of $V(H_{q+1})$ is bounded by $O(|V(H_{q+1})|) \leq O(\hat{W})$.

Next, we compute the set $\hat{\Gamma}_{q+1} = \bigcup_{e \in E_{q+1}} \hat{\Gamma}(e)$ of edges in graph $X' \setminus F$, where for an edge $e \in E_{q+1}$, $\hat{\Gamma}(e) = \bigcup_{e' \in \Gamma(e)} S(e')$. In other words, an edge $\hat{e} \in E(X') \setminus F$ belongs to the set $\hat{\Gamma}(e)$ if and only if its corresponding embedding path $Q(\hat{e})$ contains at least one edge of $\Gamma(e)$, or, equivalently, edge $\hat{e}$ belongs to set $S(e')$ of edges for some edge $e' \in \Gamma(e)$. From Observation 7.17, for every edge $e \in E_{q+1}$, $|\hat{\Gamma}(e)| \leq 2\eta''$. Therefore, $|\hat{\Gamma}_{q+1}| \leq 2|E_{q+1}| \cdot \eta'' \leq 2\varphi \cdot z_{q+1} \cdot \eta'' \leq O(\varphi \cdot z_{q+1} \cdot (\log N)^{13})$. Moreover, given an edge $e \in E_{q+1}$, set $\hat{\Gamma}(e)$ can be computed in time $O(\hat{W} \cdot \eta'')$. Indeed, recall that, from the proof of Observation 7.17, an edge $\hat{e} \in E(X) \setminus F$ may lie in $\hat{\Gamma}(e)$ in one of the following two cases: (i) an endpoint of $\hat{e}$ lies in $T \cap P^*(e)$ – all such edges $\hat{e}$ can be identified in time $O(|E(P^*(e))|) \leq O(\hat{W})$; or (ii) if e' is the first edge on $P^*(e)$ and $\hat{e} \in S(e')$ – since $|S(e')| \leq \eta''$, all such edges can be identified in time $O(\eta'')$. Therefore, the time required to compute the set $\hat{\Gamma}_{q+1}$ of vertices is bounded by $O(|E_{q+1}| \cdot \hat{W} \cdot \eta'') \leq O(\varphi \cdot z_{q+1} \cdot \hat{W} \cdot \eta'') \leq O(\varphi \cdot z_{q+1} \cdot \hat{W} \cdot (\log N)^{13})$.

We also compute a partition $(T_{q+1}, \overline{T}_{q+1})$ of the vertices of X', where $T_{q+1} = V(X') \cap Z^*_{q+1}$ and $\overline{T}_{q+1} = V(X') \cap \overline{Z}^*_{q+1}$. Notice that the running time of the procedure so far is bounded by:

$$O\left(\varphi z_{q+1} \cdot (\log N)^{13} \cdot \hat{W}\right) + O(\hat{W}) \leq O\left(\frac{\hat{W} \cdot z_{q+1} \cdot (\log N)^{13}}{d'}\right) + \leq O(\hat{W} \cdot z_{q+1}),$$

since $d' \geq \frac{d}{(\log N)^{28}}$, and $d \geq (\log N)^{64}$ from the definition of the problem.

Next, we consider two cases. The first case happens if $|T_{q+1}| \geq |T| \cdot \left(1 - \frac{8}{\log^4 N}\right)$. In this case we output the cut (A_{q+1}, B_{q+1}), and then update our data structures accordingly: we delete the vertices of $\overline{Z}_{q+1}$ from G, and we delete the vertices of $\overline{Z}^*_{q+1}$ from H. We also delete the edges of $\hat{\Gamma}_{q+1}$ and the vertices of $\overline{T}_{q+1}$ from X and from X'. Finally, we update the algorithms that maintain data structures DS^{in} and DS^{out} with the vertices that were just deleted from X. In this case, the running time of the procedure remains at most $O(\hat{W} \cdot z_{q+1})$. Notice that, following this procedure, at least z_{q+1} vertices are deleted from H. We *charge* the running time of the procedure to the vertices that are deleted from H, where the charge to each vertex is $O(\hat{W})$.

We now consider the case where $|T_{q+1}| < |T| \cdot \left(1 - \frac{8}{\log^4 N}\right)$. In this case, we apply the algorithm from Claim 7.19 to graph G_0 and the sequence $(A_1, B_1), \ldots, (A_{q+1}, B_{q+1})$ of cuts in G. Recall that the running time of the algorithm is $O(\hat{W}^2)$, and it returns a finalizing cut $(\tilde{A}, \tilde{B})$ in the current graph H. Next, we apply the algorithm from Claim 7.8 to graph G_0, the sequence $(A_1, B_1), \ldots, (A_q, B_q)$ of cuts in G, and the finalizing cut $(\tilde{A}, \tilde{B})$ in the current graph H. Recall that the running time of the algorithm from Claim 7.8 is $O(\hat{W}^2)$. If the algorithm from Claim 7.8 estabishes that at least one of the events $\mathcal{E}_2, \mathcal{E}_3$ has happened, then we terminate the current phase and return "FAIL". Otherwise, we obtain a strongly well-structured φ-sparse cut (A^*, B^*) in the current graph G, such that $w(A^*), w(B^*) \geq \frac{\hat{W}}{\log^{30} N}$. Let $Z^* \in \{A^*, B^*\}$ be the side of smaller weight; if $w(A^*) = w(B^*)$, then we let $Z^* = B^*$. We delete the vertices of Z^* from G, together with the endpoints of the edges in $E_G(A^*, B^*)$. Every vertex v that was deleted from G is also deleted from H, and for every special edge e that was deleted from G, all vertices and edges on path $P^*(e)$ are deleted from H. As the result, at least $\frac{\hat{W}}{\log^{30} N}$ vertices are deleted from H and the current phase terminates.

Overall, the last execution of Procedure $\mathsf{ProcessCut}'$ in a phase takes time $O(\hat{W}^2)$. If q^* is the total number of times that the procedure is executed during the phase, then all but the last execution of the procedure take time at most:

$$\sum_{q'=1}^{q^*-1} O(z_{q'} \cdot \hat{W})) \leq O((W^0)^2).$$

(Here, we used the fact that, from Observation 7.2, the total number of vertices that ever belonged to H is bounded by $O(W^0)$.) The total running time of all executions of Procedure $\mathsf{ProcessCut}'$ over the course of a phase is then bounded by $O(\hat{W}^2)$, and the total running time of a single phase of the algorithm for the $\mathsf{MaintainCluster}$ problem remains at most:

$$O\left(c_1 \cdot (W^0)^2 \cdot 2^{c_2\sqrt{\log N}} \cdot (\log N)^{16c_2(r-1)+8c_2+30}\right) + O\left(\frac{W^0 \cdot \Delta}{(\log N)^{30}}\right).$$

7.7.5 Maintaining the Additional Data Structures and the Final Accounting

Additional Data Structures. It remains to show an algorithm that maintains, for every edge $e \in E(H)$, the set $S(e) \subseteq E(X) \setminus F$ of edges. It is straightforward to augment our algorithm to maintain this data structure without increasing its asymptotic running time.

Once the algorithm from Lemma 7.11 computes the expander X and the embedding $\mathcal{Q}$ of $X \setminus F$ into H, we initialize the sets $S(e)$ for all edges $e \in E(H)$ in a straightforward way: we start by setting $S(e) = \emptyset$ for all $e \in E(H)$. We then consider every edge $\hat{e} \in E(X) \setminus F$ one by one. When edge $\hat{e}$ is considered, we process every edge $e \in E(Q(\hat{e}))$, and add $\hat{e}$ to the corresponding set $S(e)$ of edges. Whenever some edge $\hat{e} \in E(X) \setminus F$ is deleted from X, we similarly process every edge $e \in E(Q(\hat{e}))$, and we delete $\hat{e}$ from set $S(e)$. Lastly, when an edge $e \in E(H)$ undergoes splitting, and new special edges e' and e'' are inserted into graph H as the result of this split, for every edge $\hat{e} \in S(e)$, we update the path $Q(\hat{e})$ with the corresponding edge split, and we insert $\hat{e}$ into the lists $S(e')$ and $S(e'')$. The updates to the lists $\{S(e)\}_{e \in E(H)}$ due to edge splitting were already taken into account in the running time analysis of the algorithm that processes post-query updates. The additional time that is required in order to maintain the lists $\{S(e)\}_{e \in E(H)}$ is asymptotically bounded by the time required to initialize these lists at the beginning of the phase, which, in turn, is asymptotically bounded by the running time of the algorithm from Lemma 7.11.

Failure probability and running time analysis. Recall that our algorithm may return "FAIL" in the following cases: (i) Event $\mathcal{E}$ has happened – the probability for that is bounded by 1/16; or (ii) the algorithm from Lemma 7.11 for constructing the expander X returns "FAIL": the probability for that is bounded by $\frac{1}{N^7}$; or (iii) the algorithm for the $\mathsf{ConnectToCenters}$ problem, that is used in order to maintain data structures DS^{in} and DS^{out} returns "FAIL": this can happen with probability at most $\frac{1}{N^7}$ for each of the two instantiations of the algorithm. Therefore, overall, the probability that the algorithm for a single phase terminates with a "FAIL" is bounded by $\frac{1}{2}$. From our analysis in Section 7.1, the number of phases in the algorithm is bounded by $O\left((\log N)^{33}\right)$. Since the running time of every phase is bounded by:

$$O\left(c_1\cdot (W^0)^2\cdot 2^{c_2\sqrt{\log N}}\cdot (\log N)^{16c_2(r-1)+8c_2+30}\right)+O\left(\frac{W^0\cdot\Delta}{(\log N)^{30}}\right),$$

using the fact that c_2 is a large enough constant and N is sufficiently large, we get that the total running time of the algorithm is bounded by:

$$\begin{aligned}O\left(c_1\cdot (W^0)^2\cdot 2^{c_2\sqrt{\log N}}\cdot (\log N)^{16c_2(r-1)+8c_2+63}\right)&+O\left(W^0\cdot\Delta\cdot\log^3 N\right)\\ &\leq c_1\cdot (W^0)^2\cdot 2^{c_2\sqrt{\log N}}\cdot (\log N)^{16c_2 r}+c_1\cdot W^0\cdot\Delta\cdot\log^4 N.\end{aligned}$$

This completes the proof of Theorem 4.12.

A Proofs Omitted from Section 4

A.1 Proof of Observation 4.4

Consider first any edge $e=(u,v)\in E(G)$ with $u\in A$ and $v\in B$. We claim that e must be a special edge. Indeed, assume otherwise, that is, e is a regular edge. Then $u\in A'$ and $v\in B'$ must hold. Moreover, edge $e=(u,v)$ lies in graph G^+, so $e\in E_{G^+}(A',B')$. Since cut (A',B') in G^+ is weakly well-structured, this is impossible, a contradiction. We conclude that every edge $e\in E_G(A,B)$ must be a special edge, and so cut (A,B) is a weakly well-structured cut in G.

Next, we prove that $w(A)\geq \frac{|A'|}{8}$; the proof that $w(B)\geq \frac{|B'|}{8}$ is symmetric. We partition the vertices of A' into three subsets. The first subset is $A'_1=A$. If vertex $v\in A'$ does not lie in A, then there is some special edge $e=(x,y)\in E(G)$, such that v is an inner vertex on path $P(e)$. If any of the endpoints x,y of e lie in A, then we add vertex v to the second set A'_2, and otherwise it is added to the third set, A'_3.

Clearly, $w(A)\geq |A|\geq |A'_1|$, and it is easy to verify that $w(A)\geq \frac{|A'_2|}{2}$ holds as well, since the weight of every vertex x in G is equal to the length of the unique special edge e that is incident to x, which, in turn, is equal to half the number of vertices on the corresponding path $P(e)$. Therefore, $w(A)\geq \frac{|A'_1|+|A'_2|}{4}$ must hold. Next, we show that $|A'_3|\leq \frac{|A'|}{2}$.

Indeed, let $\tilde E$ be the collection of all special edges $e=(x,y)$ in G with $x,y\in B$, such that at least one inner vertex on the corresponding path $P(e)$ belongs to A'. Clearly, for each such edge e, at least one edge $e'\in P(e)$ must belong to $E_{G^+}(A',B')$. Therefore, $|\tilde E|\leq E_{G^+}(A',B')\leq \varphi\cdot |A'|$. Since we have assumed that graph G does not contain edges of length greater than $\frac{1}{4\varphi}$, we get that, for every edge $e\in \tilde E$, the number of vertices lying on the corresponding path $P(e)$ is at most $\frac{1}{2\varphi}$. Therefore, $|A'_3|\leq \frac{1}{2\varphi}\cdot|\tilde E|\leq \frac{|A'|}{2}$. We conclude that $|A'_3|\leq \frac{|A'|}{2}$, and so $w(A)\geq \frac{|A'_1|+|A'_2|}{4}\geq \frac{|A'|}{8}$. Using a similar reasoning, we get that $w(B)\geq \frac{|B'|}{8}$.

Recall that $\Phi_{G^+}(A',B')=\frac{|E_{G^+}(A',B')|}{\min\{w(A'),w(B')\}}=\frac{|E_{G^+}(A',B')|}{\min\{|A'|,|B'|\}}$, since the weight of every vertex in G^+ is 1. Consider now any edge $e=(u,v)\in E_G(A,B)$. Since (A,B) is a well-structured cut in G, e must be a special edge, and moreover, $u\in A'$, $v\in B'$ must hold. Therefore, at least one edge on the path $P(e)$ must lie in $E_{G^+}(A',B')$. We conclude that $|E_G(A,B)|\leq |E_{G^+}(A',B')|$. Altogether, we get that:

$$\Phi_G(A,B)=\frac{|E_G(A,B)|}{\min\{w(A),w(B)\}}\leq \frac{8|E_{G^+}(A',B')|}{\min\{|A'|,|B'|\}}=8\Phi_{G^+}(A',B').$$

A.2 Proof of Observation 4.5

We start with $(A',B')=(A,B)$, and then consider every vertex $x\in A$ one by one. If there is a special edge $e=(y,x)\in E_G(B',A')$, then if $w(A')\leq w(B')$ currently holds, we move y from B' to A', and otherwise we move x from A' to B'. Notice that, since graph G is well-structured, if a regular edge (a,b) lies in G, then $a\in L$ and $b\in R$ must hold. Additionally, since (y,x) is a special edge, $y\in R$ and $x\in L$ must hold. Therefore, the only edge entering x is the special edge (y,x), and all edges leaving x are regular edges. By moving x from A' to B' we therefore cannot introduce any new edges to $E_G(A',B')$, or any new special edges to $E_G(B',A')$. Similarly, the only edge leaving y is the special edge (y,x), and all edges entering y are regular edges. Therefore, by moving y from B' to A', we cannot introduce any new edges to $E_G(A',B')$, or any new special edges to $E_G(B',A')$.

Once every vertex of A is processed, we obtain the final cut (A',B'), that is strongly well-structured.

Note that $|E_G(A',B')| \leq |E_G(A,B)| \leq \frac{\varphi}{2} \cdot \min\{w(A),w(B)\}$. Let $z = \min\{w(A),w(B)\}$, and let recall that $w(V(G)) = 2\sum_{e\in E}\ell(e) = 2W$. Notice that $z \leq W$ must hold.

If no vertices were ever removed from A', then $w(A') = w(A)$ holds. Otherwise, consider the last time when a vertex was removed from A'. We denote the removed vertex by v. Then, just before v was moved from A' to B', $w(A') \geq w(B')$ held. Therefore, at the end of the algorithm, $w(A') \geq W - w(v) \geq \frac{W}{2}$, since the weight $w(v)$ of vertex v is the length of the unique special edge incident to v, which must be bounded by $\frac{W}{2}$. We conclude that, at the end of the algorithm, $w(A') \geq \min\left\{w(A), \frac{W}{2}\right\} \geq \frac{w(A)}{4}$ holds. Using similar reasoning, $w(B') \geq \min\left\{w(B), \frac{W}{2}\right\} \geq \frac{w(B)}{4}$. Moreover, $\min\{w(A'),w(B')\} \geq \min\left\{w(A),w(B),\frac{W}{2}\right\} \geq \frac{1}{2}\min\{w(A),w(B)\}$. Altogether, $|E_G(A',B')| \leq \frac{\varphi}{2} \cdot \min\{w(A),w(B)\} \leq \varphi \cdot \min\{w(A'),w(B')\}$.

The running time of the above algorithm is $O(\mathrm{Vol}(A))$. If $\mathrm{Vol}(B) < \mathrm{Vol}(A)$, then we can execute the same procedure by examining the vertices of B instead of A, in time $O(\mathrm{Vol}(B))$.

B Proofs Omitted from Section 5

B.1 Proof of Observation 5.5

From Invariant I1, graph $H[X^0]$ is a perfect well-structured graph. It is immediate to verify that the lengths $\ell(e)$ of edges $e \in E(H)$ define a proper assignment of lengths to the edges of $H[X^0]$. Recall that we have denoted by $W^0(X)$ the total weight of all vertices in $H[X^0]$ when X is added to $\mathcal{X}$, and, from Observation 5.3, we are guaranteed that $W^0(X) \leq N'$, where $N' = \max\left\{N, 8n\cdot\log^5 n\right\}$ is the parameter that we have defined. From the definition of the s-t-SP problem, we also get that $1 \leq \eta \leq \Delta$ and $1 \leq \Delta \leq n - |B^0| \leq N \leq N'$. From our assumption that N is greater than a sufficiently large constant, with $N > 2^{2^{c_2}}$, the same holds for N'.

Since X is a non-leaf cluster,$|X^0| > \frac{\Delta\cdot(\log N)^{128}}{\eta}$ must hold. Recall that $d_X = \min\left\{\frac{|X^0|\cdot\eta}{\Delta}, d\right\}$. From Inequality 5, $d \geq 2^{2\sqrt{\log N}}$. Therefore, if $d_X = d$, then $d_X \geq 2^{2\sqrt{\log N}} \geq 2^{\sqrt{\log N'}} \geq (\log N')^{64}$ (from Inequality 4, and since $N' \geq N$ is large enough). Otherwise:

$$d_X = \frac{|X^0|\cdot\eta}{\Delta} \geq (\log N)^{128} \geq (\log N')^{64},$$

since $\log N' \leq 2\log N \leq \log^2 N$ from Inequality 3. In either case, $d_X \geq (\log N')^{64}$ holds.

Lastly, since $d_X = \min\left\{\frac{|X^0|\cdot\eta}{\Delta}, d\right\}$, we get that $d_X \leq \frac{|X^0|\cdot\eta}{\Delta}$, and so $\frac{\Delta\cdot d_X}{\eta} \leq |X^0|$ holds. This establishes that $(H[X^0], \Delta, \eta, d_X, N')$ is a valid input instance for the $\mathsf{MaintainCluster}$ problem.

In order to show that this instance is r-restricted, we need to show that $r \leq \left\lceil\sqrt{\log N'}\right\rceil$ and that $d_X \leq 2^{r\cdot\sqrt{\log N'}}$ holds. The former follows immediately since, from the definition of the $(r+1)$-restricted s-t-SP problem, $r+1 \leq \left\lceil\sqrt{\log N}\right\rceil \leq \left\lceil\sqrt{\log N'}\right\rceil$. The latter also immediately follows since $d_X \leq d$, and since $d \leq 2^{r\cdot\sqrt{\log N}} \leq 2^{r\cdot\sqrt{\log N'}}$ from Inequality 5.

B.2 Proof of Observation 5.6

From Invariant I1, graph $H[X]$ is a perfect well-structured graph, so every vertex of X is incident to exactly one special edge in $H[X]$. Clearly, both $H[Y']$ and $H[Z']$ are well-structured graphs. It remains to show that each of them is a strongly well-structured graph.

We start by proving that $H[Y']$ is a strongly well-structured graph. It is enough to show that, for every vertex $v \in Y'$, if e is the unique special edge that is incident to v in G, then the other endpoint u of e also lies in Y'. Indeed, assume otherwise. Since $H[X]$ is a perfect well-structured graph, $u \in X$ must hold. It is impossible that $u \in J$, since in this case $u \in R$, and the unique special edge incident to u lies in the cut $E_H(Y, Z)$. Therefore, $u \in Z$ must hold. Since $v \notin J$, edge e must lie in $E_H(Z, Y)$. But from the definition of a strongly well-structured cut, e must be a regular edge, a contradiction.

In order to prove that $H[Z']$ is a strongly well-structured graph, it is enough to show that, for every vertex $v \in Z'$, if e is the unique special edge that is incident to v in G, then the other endpoint u of e also lies in Z'. Assume otherwise. As before, since $H[X]$ is a perfect well-structured graph, $u \in X$ must hold. It is impossible that $u \in J'$, since in this case $u \in L$, and the unique special edge incident to u lies in the cut $E_H(Y, Z)$. Therefore, $u \in Y$ must hold. Since $v \notin J'$, edge e must lie in $E_H(Z, Y)$. But from the definition of a strongly well-structured cut, e must be a regular edge, a contradiction.

B.3 Proof of Observation 5.7

Let $e = (u, v)$ be a new right-to-left edge that was added to the $\mathcal{ATO}$ as the result of the current transformation. Assume first that $v \in J'$. Since $J' \subseteq L$, edge e must be the unique special edge entering v, and, from the definition of the set J' of vertices, $e \in E_{H[X]}(Y, Z) = E_H(Y, Z)$ must hold. Therefore, $u \in J$ must hold. Assume next that $v \in Z'$. Since edge e is a right-to-left edge, it must be the case that $u \in Y' \cup J = Y$. However, there can be no edges connecting Y to Z' in $G[X]$, from the definition of the sets J' and Z' of vertices. Assume now that $v \in Y'$. Since edge e is a right-to-left edge, $u \in J$ must hold. However, as observed already, $J \subseteq R$ must hold, and for every vertex $u \in J$, the only edge that leaves u is the unique special edge incident to u, whose other endpoint is in J'. Therefore, it is impossible that $u \in J$ and $v \in Y'$ holds for an edge (u, v). Lastly, we need to consider the case where $v \in J$. Since (u, v) is a right-to-left edge, it must be the case that $u \in J$. But since $J \subseteq R$, it is impossible that an edge of G has both endpoints in J.

B.4 Proof of Observation 5.8

As observed already, graph $H[U^0]$ is a perfect well-structured graph. It is immediate to verify that the initial lengths $\ell(e)$ of edges $e \in E(H)$ define a proper assignment of lengths to the edges of $H[U^0]$. Moreover, initially, the length of every special edge of G is set to 1. Recall that $|U^0| > 2(n - |B^0|) \cdot (\log N)^{128}$, and that $|U^0 \setminus \beta| \leq n - |B^0|$ must hold. Therefore, $|\beta| \geq \frac{99|U^0|}{100}$ holds, as required. From our assumption that N is greater than a sufficiently large constant, so is N'.

Recall that we have denoted by $W^0(U)$ the total initial weight of all vertices in $H[U^0]$. It is immediate to verify that the initial weight of every vertex in $H[U^0]$ is 1, and so $W^0(U) \leq n \leq N \leq N'$. From the definition of the s-t-SP problem, we also get that $1 \leq \eta \leq \Delta$ and $1 \leq \Delta \leq n - |B^0| \leq \frac{|\beta|}{(\log N)^{128}}$ (the latter inequality follows since we have established that $|\beta| \geq \frac{99|U^0|}{100}$ and $|U^0| \geq 2(n - |B^0|) \cdot (\log N)^{128}$).

Recall that $d_U = \max\left\{(\log N')^{64}, \left(|U^0 \setminus B^0| + \Delta\right) \frac{\eta}{4\Delta \cdot 2^{\sqrt{\log N}}}\right\}$, so $d_U \geq (\log N')^{64}$ holds. Lastly, we need to establish that $\frac{\Delta \cdot d_U}{\eta} \leq |U^0 \setminus B^0| + \Delta \cdot (\log N')^{64}$. Indeed, since $d_U = \max\left\{(\log N')^{64}, \left(|U^0 \setminus B^0| + \Delta\right) \frac{\eta}{2\Delta \cdot 2^{\sqrt{\log N}}}\right\}$, we get that:

$$\frac{\Delta \cdot d_U}{\eta} = \max\left\{\frac{\Delta \cdot (\log N')^{64}}{\eta}, \frac{|U^0 \setminus B^0|}{4 \cdot 2^{\sqrt{\log N}}} + \frac{\Delta}{4 \cdot 2^{\sqrt{\log N}}}\right\} \leq |U^0 \setminus B^0| + \Delta \cdot (\log N')^{64}.$$

In order to show that this instance is r-restricted, we need to show that $r \leq \lceil\sqrt{\log N'}\rceil$ and that $d_U \leq 2^{r\cdot\sqrt{\log N'}}$ holds. The former follows immediately since, from the definition of the $(r+1)$-restricted s-t-SP problem, $r+1 \leq \lceil\sqrt{\log N}\rceil \leq \lceil\sqrt{\log N'}\rceil$. The latter also immediately follows since:

$$\begin{aligned}
d_U &= \max\left\{(\log N')^{64}, \left(|U^0 \setminus B^0| + \Delta\right) \frac{\eta}{4\Delta \cdot 2^{\sqrt{\log N}}}\right\} \\
&\leq \max\left\{(\log N')^{64}, \frac{(n - |B^0|)\cdot\eta}{2\Delta \cdot 2^{\sqrt{\log N}}}\right\} \\
&\leq \max\left\{(\log N')^{64}, 2d\right\} \\
&\leq 2^{r\sqrt{\log N'}},
\end{aligned}$$

since $d = \frac{(n-|B^0|)\cdot\eta}{4\Delta\cdot 2^{\sqrt{\log N}}}$ and $d \leq 2^{r\cdot\sqrt{\log N}-1} \leq 2^{r\sqrt{\log N'}-1}$ from Inequality 5.

B.5 Proof of Observation 5.10

Recall that, at the beginning of the algorithm, for every edge $e \in E^{\mathsf{spec}} \setminus E^*$, $\ell(e) = 1$, and for every vertex $v \in \tilde{U}$, $w(v) = 1$ holds. For every edge $e \in (E^{\mathsf{spec}} \setminus E^*) \cap H[U^0]$, if both endpoints of e lie in U at the end of the algorithm, then we set $\tilde{\ell}(e)$ to be the length of e at the end of the algorithm, and otherwise we let $\tilde{\ell}(e)$ be the length of e just before its endpoints are deleted from U. Let $E' \subseteq (E^{\mathsf{spec}} \setminus E^*) \cap H[U^0]$ be the set of all edges $e \in (E^{\mathsf{spec}} \setminus E^*) \cap H[U^0]$ with $\tilde{\ell}(e) > 1$. Then $\sum_{v\in\tilde{U}} \tilde{w}(v) \leq |\tilde{U}| + 2\sum_{e\in E'} \tilde{\ell}(e)$.

Let q be the total number of queries to algorithm $\mathcal{A}'(U)$, so that $q \leq \Delta$, and let $\mathcal{Q}' = \{P_1, \ldots, P_q\}$ be the paths returned by Algorithm $\mathcal{A}'(U)$ in response to the queries, where the paths are indexed in the order in which they were returned. For all $1 \leq i \leq q$, let C_i denote the length of path P_i when it was added to $\mathcal{Q}$. Clearly, $C_i \leq d_U$, and $\sum_{i=1}^q C_i \leq q \cdot d_U \leq \Delta \cdot d_U$.

Consider now some edge $e \in E'$, and recall that $\ell(e)$ was doubled at least once over the course of Algorithm $\mathcal{A}'(U)$. Let τ be the last time when the length of e was doubled. Then, prior to time τ, there were at least η iterations i, during which the path P_i that was added to $\mathcal{Q}'$ contained e, and the length of e during the corresponding iteration was $\frac{\tilde{\ell}(e)}{2}$. In other words, edge e contributes at least $\frac{\tilde{\ell}(e)\eta}{2}$ to $\sum_{i=1}^q C_i$. Altogether, we get that $\sum_{i=1}^q C_i \geq \sum_{e\in E'} \frac{\tilde{\ell}(e)\eta}{2}$. Therefore:

$$\sum_{e\in E'} \tilde{\ell}(e) \leq \frac{2\sum_{i=1}^q C_i}{\eta} \leq \frac{2\Delta \cdot d_U}{\eta} \leq |U^0 \setminus B^0| + 2\Delta \cdot (\log N')^{64},$$

since $d_U = \max\left\{(\log N')^{64}, \left(|U^0 \setminus B^0| + \Delta\right) \frac{\eta}{4\Delta\cdot 2^{\sqrt{\log N}}}\right\}$.

Altogether, we get that:

$$\sum_{v\in\tilde{U}} \tilde{w}(v) \leq |\tilde{U}| + 2\sum_{e\in E'} \tilde{\ell}(e) \leq 4|U^0 \setminus B^0| + 8\Delta \cdot (\log N')^{64},$$

since the set $\tilde{U}$ contains all vertices of $U^0 \setminus B^0$, and additionally all vertices of $B^0 \cap U^0$ that were deleted from B^0 over the course of the algorithm, whose number is bounded by Δ.

B.6 Proof of Claim 5.11

Consider any cluster $X \in \tilde{X}'_2$. Observe first that at most one child-cluster of X may be its close descendant. Indeed, assume otherwise, and let X_1, X_2 be two child-clusters of X that are its close descendants, and assume that X_1 was added to $\mathcal{X}$ before X_2. Then $|X_1^0|, |X_2^0| > |X^0|/2$ must hold, and so $X_1^0 \cap X_2^0 \neq \emptyset$. But this is impossible since, after cluster X_1 was added to $\mathcal{X}$, all vertices of X_1^0 were deleted from X. Recall that, if X_1 is a child-cluster of X, and it is a close descendant of X, then, when X_1 was created, $\sum_{v \in X_1} w(v) \leq \sum_{v \in X} w(v)/2$ held, following which the vertices of X_1^0 were deleted from X. We start with the following claim that bounds the number of close descendants of a cluster X, whose initial weights are not much lower than that of X.

Claim B.1 *Consider a sequence $(X_1, X_2, \ldots, X_z)$ of distinct clusters in $\tilde{\mathcal{X}}'_2$, such that for all $1 \leq i < z$, X_{i+1} is a child cluster of X_i, clusters $X_2, \ldots, X_z$ are close descendants of X_1, and for all $2 \leq i \leq z$, $W^0(X_i) \geq W^0(X_1)/2$. Then $z \leq 16$ must hold.*

Proof: For all $1 \leq i \leq z$, let τ_i be the time when cluster X_i was added to $\mathcal{X}$, and for all $1 < i \leq r$, let X'_{i-1} be the set of all vertices that lied in X_{i-1} just before time τ_i, excluding the vertices of X_i. For convenience, we also denote by $X'_z = X_z$. For all $1 \leq i < z$, we denote by W_i be the total weight of all vertices in set X'_i at time τ_{i+1}, and we denote by $W_z = W^0(X_z)$. Note that the sets $X'_1, \ldots, X'_z$ of vertices are mutually disjoint.

Consider now an index $1 < i \leq z$. Since $|X_i^0| \geq |X_1^0|/2$ must hold, we get that $|X_i^0| \geq |X_{i-1}^0|/2$, and so, from our discussion, at time τ_i, the total weight of the vertices of X_i was less than or equal to the total weight of the vertices of X'_{i-1}. In other words, for all $1 < i \leq z$, $W_{i-1} \geq W^0(X_i) \geq W^0(X_1)/2$. From our definitions, $W_z = W^0(X_z) \geq W^0(X_1)/2$ as well. Therefore:

$$\sum_{i=1}^{z} W_i \geq \frac{W^0(X_1) \cdot z}{2}.$$

In the following observation, we show that $\sum_{i=1}^{z} W_i \leq 8W^0(X_1)$ must hold, and so $z \leq 16$, as required. The following observation then completes the proof of Claim B.1.

Observation B.2 $\sum_{i=1}^{z} W_i \leq 8W^0(X_1)$.

Proof: Recall that, from our definition, the sets $\{X'_1, \ldots, X'_z\}$ of vertices are mutually disjoint. For convenience, denote $\tau_{z+1} = \tau_z$, and let $Y = \bigcup_{i=1}^{z} X'_i$, so $Y \subseteq X_1^0$. Recall that, for all $1 \leq i < z$, W_i denotes the weight of the vertices in X'_i at time τ_{i+1}. In order to prove the observation, we will observe how the weights of the vertices of Y evolve during the time interval $(\tau_1, \tau_z]$, except that, for all $1 \leq i \leq z$, we will "freeze" the weights of the vertices in X'_i at time τ_{i+1}, so they are not allowed to grow anymore.

Specifically, for every vertex $v \in Y$, we denote by $\tilde{w}(v)$ the weight of v at time τ_1. For all $1 \leq i \leq z$, for every vertex $v \in X_i$, we denote by $\tilde{w}'(v)$ the weight of v at time τ_{i+1}. Clearly, $\sum_{v \in Y} \tilde{w}(v) \leq W^0(X_1)$, and $\sum_{i=1}^{z} W_i = \sum_{v \in Y} \tilde{w}'(v)$. It now remains to bound $\sum_{v \in Y} (\tilde{w}'(v) - \tilde{w}(v))$. Let $\tilde{E}'$ be the set of all special edges that lie in $E(H[X_1^0])$. For every special edge $e \in \tilde{E}'$, we denote by $\tilde{\ell}(e)$ the length of e at time τ_1, and recall that $W^0(X_1) = 2 \sum_{e \in \tilde{E}'} \tilde{\ell}(e)$.

Recall that for each cluster $X \in \tilde{\mathcal{X}}'_2$, we have defined a parameter $d_X = \min \left\{ \frac{|X^0| \cdot \eta}{\Delta}, d \right\}$. Since, for all $1 < i \leq z$, cluster X_i is a descendant of X_1, we get that $|X_i^0| \leq |X^0|$, and so $d_{X_i} \leq d_{X_1}$.

Consider now the following process. Initially, every vertex $v \in Y$ is assigned weight $\tilde{w}(v)$, and every special edge $e \in \tilde{E}'$ is assigned length $\tilde{\ell}(e)$. Then we obeserve the iterations of the algorithm that occured during the time interval $[\tau_1, \tau_z)$ in the order in which they occur. Let j be any such iteration. Let $1 \leq i \leq z$ be the integer for which iteration j occurs during the time interval $[\tau_i, \tau_{i+1})$. We then consider the cluster X_i, and denote by P_j the path that Algorithm $\mathcal{A}(X_i)$ returned in response to a short-path query during iteration j. Denote by C_j the length of P_j at the time when it was returned, and recall that $C_j \leq d_{X_i} \leq d_{X_1}$ must hold. We also denote by $\tilde{E}_j \subseteq E(P_j)$ the set of special edges of $H[X_i]$, whose lengths were doubled as the result of this iteration. For every vertex $v \in X_i \cap Y$, that serves as an endpoint of an edge of $\tilde{E}_j$, we double the weight $\tilde{w}(v)$. For every edge $e \in \tilde{E}_j$, we also double its length $\tilde{\ell}(e)$.

It is easy to verify that, at the end of this process, the final weight of every vertex $v \in Y$ is precisely $\tilde{w}'(v)$. For every special edge $e \in \tilde{E}'$, let $\tilde{\ell}'(e)$ be the length of e at the end of this process. Then $\sum_{v \in Y} \tilde{w}'(v) \leq 2 \sum_{e \in \tilde{E}'} \tilde{\ell}'(e)$. Let r be the total number of iterations of the algorithm that occur during the time interval $[\tau_1, \tau_z)$. Consider now any special edge $e \in \tilde{E}'$. From the definition of our algorithm, if $\tau_1 \leq \tau < \tau' \leq \tau_z$ are any two times at which the length of e was doubled, then edge e participated in at least η of the paths of $\{P_1, \ldots, P_r\}$ that were returned by the above process during the time interval $[\tau, \tau')$.

Recall that we have established that $\sum_{e \in \tilde{E}'} \tilde{\ell}(e) = \frac{W^0(X_1)}{2}$. Let $\tilde{E}'' \subseteq \tilde{E}'$ be the set of all edges e for which $\tilde{\ell}'(e) > 2\tilde{\ell}(e)$. Consider now any such edge $e \in \tilde{E}''$. Then the length of e must have been doubled at least twice during the process that we described above. Let $\tau < \tau'$ be the last two times during the process when the length of e was doubled. Then, at least η of the paths of $\{P_1, \ldots, P_r\}$ that were returned during the time interval $[\tau, \tau')$ contained e, and for each such path P_j, edge e contributed at least $\tilde{\ell}'(e)/2$ to the length of the path. Therefore, every edge $e \in \tilde{E}''$ contributes at least $\frac{\tilde{\ell}'(e) \cdot \eta}{2}$ to $\sum_{j=1}^r C_j$, and so $\sum_{j=1}^r C_j \geq \sum_{e \in \tilde{E}''} \frac{\tilde{\ell}'(e) \cdot \eta}{2}$. On the other hand:

$$\sum_{j=1}^{r} C_j \leq r \cdot d_{X_1} \leq \Delta \cdot d_{X_1} \leq \Delta \cdot \min\left\{\frac{|X_1^0| \cdot \eta}{\Delta}, d\right\} \leq |X_1^0| \cdot \eta \leq W^0(X_1) \cdot \eta.$$

Altogether, we get that $\sum_{e \in \tilde{E}''} \tilde{\ell}'(e) \leq \frac{2}{\eta} \cdot \sum_{j=1}^r C_j \leq 2W^0(X_1)$, and so:

$$\sum_{i=1}^{z} W_i = \sum_{v \in Y} \tilde{w}'(v) \leq 2 \sum_{e \in \tilde{E}'} \tilde{\ell}'(e) \leq 4 \sum_{e \in \tilde{E}'} \tilde{\ell}(e) + 2 \sum_{e \in \tilde{E}''} \tilde{\ell}'(e) \leq 8W^0(X_1).$$

□ □

Recall that, for every cluster $X \in \tilde{\mathcal{X}}_2'$, $\frac{\Delta \cdot (\log N)^{128}}{\eta} \leq W^0(X) \leq 8n \log^5 n$ holds (see the definition of leaf clusters (Definition 5.4) and Inequality 6). We now prove that, for all $\left\lfloor \log\left(\frac{\Delta \cdot (\log N)^{128}}{\eta}\right) \right\rfloor < j \leq 4 \lceil \log n \rceil$, if $X \in \tilde{\mathcal{X}}_2'$ is a cluster with $2^j \leq W^0(X) < 2^{j+1}$, then X has at most $16j$ close descendants.

The proof is by induction on j. The base case is when $j = \left\lfloor \log\left(\frac{\Delta \cdot (\log N)^{128}}{\eta}\right) \right\rfloor$. Recall that, if $W^0(X') < \frac{\Delta \cdot (\log N)^{128}}{\eta}$, then X' is a leaf cluster. Therefore, if $X \in \tilde{\mathcal{X}}_2'$ is a cluster with $2^j \leq W^0(X) < 2^{j+1}$, and $X' \in \tilde{\mathcal{X}}_2$ is its close descendant, then $W^0(X') \geq \frac{\Delta \cdot (\log N)^{128}}{\eta} \geq \frac{W^0(X)}{2}$ must hold. For a cluster $X \in \tilde{\mathcal{X}}_2'$ with $2^j \leq W^0(X) < 2^{j+1}$, let $\mathcal{D}(X) \subseteq \tilde{X}_2'$ be the collection of the close descendants of X, and denote $|\mathcal{D}(X)| = z$. Then we can define an ordering $(X = X_1, X_2, \ldots, X_z)$ of the clusters in $\mathcal{D}(X)$, such that, for all $1 \leq i < z$, X_{i+1} is a child-cluster of X_i. From Claim B.1, we are then guaranteed that $z \leq 16$.

For the induction step, assume that the claim holds for some integer $j \geq \left\lfloor \log\left(\frac{\Delta\cdot(\log N)^{128}}{\eta}\right)\right\rfloor$, and consider a cluster $X \in \tilde{\mathcal{X}}'_2$ with $2^{j+1} \leq W^0(X) < 2^{j+2}$. Let $\mathcal{D}(X) \subseteq \tilde{X}'_2$ be the set of all close descendants of X, and denote $|\mathcal{D}(X)| = z$. As before, we can define an ordering $(X = X_1, X_2, \ldots, X_z)$ of the clusters in $\mathcal{D}(X)$, such that, for all $1 \leq i < r$, X_{i+1} is a child-cluster of X_i. Let $i' > 1$ be the smallest index with $W^0(X_{i'}) < W^0(X)/2$. Then for all $1 \leq i < i'$, $W^0(X_i) \geq W^0(X)/2$, and so, from Claim B.1, $i' \leq 17$. Moreover, clusters $X_{i'+1}, \ldots, X_z$ are close descendants of cluster $X_{i'}$, and so, from the induction hypothesis, $z - i' + 1 \leq 16j$. Altogether, we get that $z = (z - i' + 1) + i' - 1 \leq 16(j+1)$. Since, for every cluster $X \in \tilde{\mathcal{X}}'_2$, $W^0(X) \leq 8n\log^5 n < 2^{4\lceil \log n\rceil}$, we get that each such cluster has at most $64\lceil \log n\rceil$ close descendants.

C Proofs Omitted from Section 6

C.1 Proof of Claim 6.2

Observe first that, from our definition, graph H' is a well-structured graph.

Recall that $N \geq \frac{W}{32}$, while $|V(H')| \leq |V(H)| + \beta \cdot M \leq 2W\log^3 N$. Since $N' = 64N\log^3 N$, $N' \geq |V(H')|$ holds. Since we have assumed that $k \geq 2^{20} \cdot d' \cdot \log^6 N$, we get that $1 \leq \eta' \leq \Delta'$, as required. Since $U_\lambda \subseteq A'$, while B' contains all vertices of U_0 and their copies, it is easy to verify that $|A'|, |B'| \geq k = \Delta'$. Therefore, for any pair (A', B') of disjoint subsets of $V(H')$ with $U_\lambda \subseteq A$, and with all vertices of U_0 and their copies lying in B', $(H', A', B', \Delta', \eta', N')$ defines a valid instance of the RouteAndCut problem. We now verify that this instance is r-restricted.

In order to do so, it is enough to show that $1 \leq r \leq \left\lceil\sqrt{\log N'}\right\rceil$ and $\frac{(|V(H')|-|B'|)\cdot\eta'}{\Delta'} \leq 2^{r\cdot\sqrt{\log N'}}$ holds. Since $1 \leq r \leq \sqrt{\log N}$, we get that $1 \leq r \leq \left\lceil\sqrt{\log N'}\right\rceil$ also holds.

Recall that $|V(H') \setminus B'| \leq 4\lambda k$. Therefore:

$$\frac{(|V(H')| - |B'|)\cdot\eta'}{\Delta'} \leq \frac{4\lambda k \cdot 2^{20}\cdot d' \cdot \log^6 N}{k} \leq d' \cdot \log^8 N \leq 2^{r\sqrt{\log N}} \leq 2^{r\cdot\sqrt{\log N'}},$$

since $\lambda = \lceil \log(64N)\rceil + 1$ and $d' \leq \frac{2^{r\sqrt{\log N}}}{\log^8 N}$ from the definition of Procedure ReconnectLayer.

C.2 Proof of Observation 6.4

We first prove that $|\tilde{Y}| \leq O(k\log N)$. Recall that set $\tilde{Y}$ may contain all vertices of H', except for the vertices of B'_j. It is easy to verify that $|B'_j| \geq |B_j| - k$, since $|\mathcal{R}'_j| < k$. Recall that we have shown that $|V(H') \setminus B_j| \leq O(\lambda\cdot k) \leq O(k\log N)$. Therefore, $|\tilde{Y}| \leq |V(H')\setminus B_j| + |B_j \setminus B'_j| \leq O(k\log N)$.

In order to prove the remaining two assertions, we partition the set β of vertices into three subsets. The first set, β_1, contains all vertices $x \in \beta$ that lie in $\tilde{Y}'$. The second set, β_2, contains all vertices $x \in \beta$ that lie in $\tilde{Y}$, such that every copy of x lies in $\tilde{Y}$ as well. The third set, β_3, contains all remaining vertices of β.

Notice that, for every vertex $x \in \beta_3$, some edge that is incident to one of the copies of x must lie in $E_{H'}(\tilde{Y}, \tilde{Y}')$, and so $|\beta_3| \leq |E_{H'}(\tilde{Y}, \tilde{Y}')| \leq \frac{k}{d'\log^6 N} \leq \frac{|\beta|}{\log^4 N}$, since $|\beta| \geq \frac{W}{d'\cdot\log^2 N} \geq \frac{k}{d'\cdot\log^2 N}$.

Notice also that, for every vertex $x \in \beta_2$, all M copies of x (including x itself) must serve as endpoints of the paths in $\mathcal{R}'_j$, so $|\beta_2| \leq \frac{|\mathcal{R}'_j|}{M} \leq \frac{k\cdot|\beta|}{W\log^3 N} \leq \min\left\{\frac{k}{\log^3 N}, \frac{|\beta|}{\log^3 N}\right\}$, since $M = \frac{W\cdot\log^3 N}{|\beta|}$ and $k, |\beta| \leq |V(H)| \leq W$.

Overall, we get that:

$$|\tilde{Y}' \cap \beta| = |\beta_1| \geq |\beta| - |\beta_2| - |\beta_3| \geq |\beta| \cdot \left(1 - \frac{1}{\log^4 N} - \frac{1}{\log^3 N}\right) \geq |\beta| \cdot \left(1 - \frac{2}{\log^3 N}\right).$$

Additionally,

$$|\tilde{Y}' \cap \beta| \geq |\beta| - |\beta_2| - |\beta_3| \geq |\beta| - \frac{k}{\log^3 N} - \frac{k}{d' \log^6 N} \geq |\beta| - \frac{2k}{\log^3 N}.$$

Lastly, since $|A'_j| \geq \frac{k}{64\lambda} \geq \frac{k}{2^9 \log N}$, as $\lambda = \lceil \log(64N) \rceil + 1 \leq 8 \log N$, while $|\beta_2| \leq \frac{k}{\log^3 N}$ and $|\beta_3| \leq \frac{k}{d' \log^6 N}$, we get that $|A'_j| \geq (|\beta_2| + |\beta_3|) \cdot \log N = |\tilde{Y} \cap \beta| \cdot \log N$.

C.3 Proof of Claim 6.12

Let β'_1 be the set of vertices that were deleted from β directly, and let $\beta'_2 = \beta' \setminus \beta'_1$. Since at most one vertex may be deleted from β directly in every iteration, $|\beta'_1| \leq \Delta$ must hold.

Let E' denote the set of all special edges whose lengths have increased over the course of the algorithm. Consider any such edge $e = (u, v) \in E'$, and let τ be the first time when the length of e was increased. Then at time τ, neither of the vertices u, v may lie in β. Therefore, either both of these vertices lied in $V(G^0) \setminus \beta^0$; or at least one of these vertices lies in β'_1 (since, once a vertex is added to β'_2, it is deleted from G). We conclude that $|E'| \leq |V(G^0)| - |\beta^0 \setminus \beta'_1| \leq n - |\beta^0| + \Delta$. For every edge $e \in E'$, let $\ell^0(e) = 1$ be its initial length. Clearly, $\sum_{e \in E'} \ell^0(e) \leq n - |\beta^0| + |\beta'_1| \leq n - |\beta^0| + \Delta$.

Denote by $\{P_1, \dots, P_q\}$ the set of all paths that the algorithm returned in response to the queries. For all $1 \leq i \leq q$, let C_i be the length of path P_i at the time it was returned, so $C_i \leq d$, and $\sum_{i=1}^q C_i \leq d \cdot q \leq d \cdot \Delta$.

Let $\hat{E} \subseteq E'$ be the set of edges whose length increased more than once. For every edge $e \in \hat{E}$, let $\ell^*(e)$ be the final length of the edge e, after its last increase, and let $\tau'(e) < \tau''(e)$ be the times of the last two increases in the length of e. Then during the time interval $(\tau'(e), \tau''(e)]$, at least η paths that were returned in response to the queries contained e, and edge e contributed at least $\frac{\ell^*(e)}{2}$ to the length C_i of each such path P_i. In other words, every edge $e \in \hat{E}$ contributes at least $\frac{\eta \cdot \ell^*(e)}{2}$ to $\sum_{i=1}^q C_i$. Therefore:

$$\sum_{i=1}^q C_i \geq \frac{\eta}{2} \sum_{e \in \hat{E}} \ell^*(e),$$

and:

$$\sum_{e \in \hat{E}} \ell^*(e) \leq \frac{2}{\eta} \cdot \sum_{i=1}^q C_i \leq \frac{2d\Delta}{\eta}.$$

Altogether, we get that:

$$\begin{aligned}\Lambda &\leq \sum_{e\in E'} 2\ell^0(e) + \sum_{e\in \hat{E}} \ell^*(e)\\ &\leq 2n - 2|\beta^0| + 2\Delta + \frac{2d\Delta}{\eta}\\ &\leq 4n - 4|\beta^0| + 3\Delta\cdot(\log N)^{64}.\end{aligned}$$

(we have used the fact that, from the problem definition, $\frac{\Delta\cdot d}{\eta} \leq |V(G)| - |\beta^0| + \Delta\cdot(\log N)^{64}$). Since, from the problem definition, $\Delta \leq \frac{|\beta^0|}{(\log N)^{128}}$, we get that $\Lambda \leq 4n - 4|\beta^0| + 3\Delta\cdot(\log N)^{64} \leq 4n$.

Next, we bound $|\beta'_2|$. Denote $\hat{\beta} = \beta^0 \setminus \beta'_1$, and recall that $|\hat{\beta}| \geq |\beta^0| - \Delta$. For every vertex $v \in V(G^0)$, let $w^0(v) = 1$ denote the initial weight of v, and let $w^*(v)$ denote the weight of v at the end of the algorithm (if $v \in V(G)$ holds), or just before v is deleted from G (otherwise). We also denote by $\tilde{W} = \sum_{v\in V(G^0)\setminus\hat{\beta}} w^*(v)$.

Notice that, at the beginning of the algorithm:

$$\sum_{v\in V(G^0)\setminus\hat{\beta}} w^0(v) = n - |\hat{\beta}| \leq n - |\beta^0| + \Delta.$$

Since the total increase in all edge lengths is $\Lambda \leq 4n - 4|\beta^0| + 5\Delta\cdot(\log N)^{64}$, we get that:

$$\tilde{W} \leq \sum_{v\in V(G^0)\setminus\hat{\beta}} w^0(v) + 2\Lambda \leq 9n - 9|\beta^0| + 11\Delta\cdot(\log N)^{64}.$$

Consider now any strongly well-structured cut (X,Y) in G of sparsity $\Phi_G(X,Y) \leq \frac{(\log N)^{64}}{d}$, such that $w(X) \geq 1.8|\beta\cap X|$ holds, that the algorithm produces. Let $J' \subseteq Y$ denote the set of vertices that serve as endpoints of the edges of $E_G(X,Y)$, and recall that $|J'| \leq |E_G(X,Y)| \leq \frac{w(X)\cdot(\log N)^{64}}{d} \leq w(X)$, since $d \geq (\log N)^{64}$ from the problem definition. Note that $\hat{\beta}\cap X \subseteq \beta\cap X$, and so $w(X) \geq 1.8|\beta\cap X| \geq 1.8|\hat{\beta}\cap X|$ holds. Since the weights of all vertices in $\hat{\beta}$ are 1, we get that:

$$w(X\setminus\hat{\beta}) = w(X) - |X\cap\hat{\beta}| \geq w(X)\left(1 - \frac{1}{1.8}\right) \geq 0.44w(X).$$

The number of vertices that were indirectly deleted due to the cut is bounded by:

$$|X\cap\hat{\beta}| + |J'| \leq 2w(X) \leq 5w(X\setminus\hat{\beta}) \leq 5\sum_{v\in X\setminus\hat{\beta}} w^*(v).$$

We assign to every vertex $v \in X\setminus\hat{\beta}$ a *charge* of $5w^*(v)$ units, so that the total charge to all vertices of $X\setminus\hat{\beta}$ is greater than the number of vertices that are deleted from β in the current iteration. Notice that the vertices of $X\setminus\hat{\beta}$ are deleted from G, and so they will never be charged again. We conclude that:

$$|\beta'_2| \leq 5\hat{W} \leq 45n - 45|\beta^0| + 55\Delta\cdot(\log N)^{64}.$$

Altogether, we get that $|\beta'| \leq |\beta'_1| + |\beta'_2| \leq 45n - 45|\beta^0| + 56\Delta\cdot(\log N)^{64}$.

C.4 Proof of Corollary 6.13

First, observe that:

$$\sum_{v\in V(G)\setminus\beta} w(v) \le |V(G)| - |\beta| + 2\Lambda \le |V(G)| - |\beta| + 8(n - |\beta^0|) + 10\Delta(\log N)^{64},$$

from Claim 6.12. Notice that, if $v \in V(G)\setminus\beta$, then either $v \in V(G^0)\setminus\beta^0$; or v belonged to β, but was directly removed from β. Since the number of vertices that are directly removed from β is bounded by Δ, and since, from the problem definition, $\Delta \le \frac{|\beta^0|}{(\log N)^{128}}$, we get that:

$$\sum_{v\in V(G)\setminus\beta} w(v) \le 9(n - |\beta^0|) + 11\Delta(\log N)^{64} \le 9(n - |\beta^0|) + \frac{|\beta^0|}{1000}.$$

Lastly, since $|\beta^0| \ge \frac{99n}{100}$ from the problem definition, we get that $(n - |\beta^0|) \le \frac{n}{100} \le \frac{|\beta^0|}{99}$. Using the fact that $|\beta| \ge \frac{|\beta^0|}{2}$ holds throughout the algorithm, we now get that:

$$\sum_{v\in V(G)\setminus\beta} w(v) \le \frac{9|\beta^0|}{99} + \frac{|\beta^0|}{1000} \le 0.1|\beta^0| \le 0.2|\beta|.$$

C.5 Proof of Observation 6.14

First, from Corollary 6.13, $w(A\setminus\beta) \le \sum_{v\in V(G)\setminus\beta} w(v) \le 0.2|\beta|$.

On the other hand, $w(A\setminus\beta) = w(A) - w(A\cap\beta) = w(A) - |A\cap\beta| \ge w(A)\left(1 - \frac{1}{1.8}\right) \ge 0.44w(A)$. Therefore, $w(A) \le 2.27w(A\setminus\beta) \le 0.46|\beta|$. In particular, $|\beta\cap A| \le w(A) \le 0.46|\beta|$. but then:

$$w(B) \ge |B\cap\beta| \ge |\beta| - |A\cap\beta| \ge 0.54|\beta| > w(A).$$

D Proofs Omitted from Section 7

D.1 Proof of Observation 7.2

Let $\mathcal{P}$ be the set of all paths that the algorithm returned in response to queries. For every path $P \in \mathcal{P}$, we denote by $\ell'(P)$ the length of the path P at the time when it was returned, so $\ell'(P) \le d$. Since $|\mathcal{P}| \le \Delta$, we get that $\sum_{P\in\mathcal{P}} \ell'(P) \le d\cdot\Delta$.

Let $E' \subseteq E^{\mathsf{spec}}$ be the set of all edges e with $\ell^*(e) > 2\ell^0(e)$. Consider any such edge $e \in E'$, and denote $\ell^*(e) = 2^i$. Let τ be the first time when the length of e was set to 2^{i-1}, and let τ' be the time when the length of e was set to 2^i. Then, in time interval $(\tau, \tau']$, at least η paths were added to $\mathcal{P}$ that contained e, and, for each such path P, edge e contributed at least $2^{i-1} = \frac{\ell^*(e)}{2}$ to $\ell'(P)$. Therefore:

$$\sum_{P\in\mathcal{P}} \ell'(P) \ge \sum_{e\in E'} \frac{\ell^*(e)\cdot\eta}{2}.$$

Since $\sum_{P\in\mathcal{P}} \ell'(P) \le d\cdot\Delta$, we get that $\sum_{e\in E'} \ell^*(e) \le \frac{2d\Delta}{\eta}$. Altogether, we get that:

$$\sum_{e\in E^{\mathsf{spec}}} \ell^*(e) \leq 2\sum_{e\in E^{\mathsf{spec}}} \ell^0(e) + \sum_{e\in E'} \ell^*(e) \leq 2\sum_{e\in E^{\mathsf{spec}}} \ell^0(e) + \frac{2d\Delta}{\eta} \leq 2\sum_{e\in E^{\mathsf{spec}}} \ell^0(e) + 2n,$$

since $\frac{\Delta d}{\eta} \leq n$ from the problem definition.

D.2 Proof of Claim 7.9

Recall that $(A_1, B_1), \ldots, (A_q, B_q)$ is the sequence of all stronlgy well-structured φ-sparse cuts that our algorithm produced during the current phase so far, indexed in the order in which the algorithm produced them. For all $1 \leq i \leq q$, denote by τ_i the time when cut (A_i, B_i) was computed, by $E_i = E_G(A_i, B_i)$, and by $J_i \subseteq A_i, J'_i \subseteq B_i$ the vertices that serve as endpoints of the edges in E_i (with respect to $G^{(\tau_i)}$ – graph G at time τ_i). If, at time τ_i, $w(A_i) > w(B_i)$, then we denote $Z_i = A_i \setminus J_i$ and $\overline{Z}_i = B_i \setminus J'_i$; otherwise, we denote $Z_i = B_i \setminus J'_i$ and $\overline{Z}_i = A_i \setminus J_i$. In the former case, we say that cut (A_i, B_i) is of *type 1*, and in the latter case it is of *type 2*.

We denote by G_0 the graph G at the beginning of the phase, and, for all $1 \leq i \leq q$, we denote by G_i the graph G that is obtained after cut (A_i, B_i) has been processed. Therefore, for all $1 \leq i \leq q$, graph G_i is obtained from graph G_{i-1} by deleting the vertices of $\overline{Z}_i \cup J_i \cup J'_i$ from it, or, equivalently, $G_i = G_{i-1}[Z_i]$. The current graph G is then denoted by G_q, and we denote the current graph H by H_q. We also let $E' = \bigcup_{j=1}^q E_j$.

Since the total number of vertices deleted so far from H during the current phase is less than $\frac{\hat{W}}{\log^{30} N}$, and since, for all $1 \leq j \leq q$, $|E_j| \leq \varphi \cdot \min\{w(A_j), w(B_j)\}$, we conclude that:

$$|E'| \leq \sum_{j=1}^q |E_j| \leq \frac{\varphi \cdot \hat{W}}{\log^{30} N} = \frac{4\hat{W}}{d' \cdot \log^{30} N} \leq \frac{8|T|}{\log^{30} N}, \tag{21}$$

since $|T| \geq \frac{\hat{W}}{2d'}$, as Event $\mathcal{E}_1$ did not happen.

Our first step is to compute a partition of the vertices of G_0 into three subsets, S_1, S_2, S_3, such that $S_2 = V(G_q)$, and the partition has several other useful properties, summarized in the following claim.

Claim D.1 *There is a deterministic algorithm, that computes a partition (S_1, S_2, S_3) of the vertices of G_0, such that the following hold:*

- *$S_2 = V(G_q)$;*
- *if $e = (x, y)$ is an edge of G_0 connecting a vertex $x \in S_j$ to a vertex $y \in S_i$ with $1 \leq j < i \leq 3$, then e is a special edge, and moreover, $e \in E'$ holds; and*
- *if $e = (x, y)$ is an edge of G_0 connecting a vertex $x \in S_j$ to a vertex $y \in S_i$ with $1 \leq i < j \leq 3$, then e is a regular edge.*

The running time of the algorithm is $O((\hat{W})^2)$.

Proof: We start by constructing a sequence $\mathcal{S} = (Y_1, \ldots, Y_{q+1})$ of subsets of vertices of G_0, over the course of q iterations. Specifically, for all $1 \leq i \leq q$, in iteration i we will construct a sequence $\mathcal{S}_i$ of $i+1$ subsets of vertices of G_0. We will also identify a single set $S^*_i \in \mathcal{S}_i$ as the *distingiushed* vertex set of $\mathcal{S}_i$, and we will define a subset $J^*_i \subseteq S^*_i$ of vertices of S^*_i called *boundary* vertices. We require that the following properties hold.

R1. The sets of vertices in $\mathcal{S}_i$ define a partition of $V(G_0)$;

R2. The set J_i^* of boundary vertices contains all vertices of S_i^* that serve as endpoints of the edges of $\bigcup_{i'=1}^{i} E_{i'}$ and only them;

R3. $V(G_i) = S_i^* \setminus J_i^*$;

R4. For every boundary vertex $v \in J_i^*$, every edge incident to v in $G_0[S_i^*]$ is a regular edge;

R5. If $S, S' \in \mathcal{S}_i$, with S appearing before S' in the sequence, and e is an edge of G_0 connecting a vertex of S' to a vertex of S, then e is a regular edge of G_0; and

R6. If $S, S' \in \mathcal{S}_i$, with S appearing before S' in the sequence, and e is an edge of G_0 connecting a vertex of S to a vertex of S', then $e \in \bigcup_{i'=1}^{i} E_{i'}$.

In the first iteration, we consider the cut (A_1, B_1), and we let the initial sequence be $\mathcal{S}_1 = (A_1, B_1)$. Notice that every edge connecting a vertex of A_1 to a vertex of B_1 is an edge of E_1, and, since cut (A_1, B_1) is strongly well-structured, every edge connecting a vertex of B_1 to a vertex of A_1 is a regular edge. This immediately establishes Properties R1, R5 and R6 for $\mathcal{S}_1$.

If (A_1, B_1) is a type-1 cut, that is, $w(A_1) > w(B_1)$, then we designate A_1 as the distinguished set S_1^* of vertices for $\mathcal{S}_1$, and we let J_1 be the corresponding set J_1^* of boundary vertices. Alternatively, if (A_1, B_1) is a type-2 cut, then we designate B_1 as the distinguished set S_1^* of vertices for $\mathcal{S}_1$, and we let J_1' be the corresponding set J_1^* of boundary vertices. It is immediate to verify that, with this definition, Properties R2–R4 hold.

Assume now that for some $1 \leq i < q$, we have computed the sequence $\mathcal{S}_i = (S_1, S_2, \ldots, S_{i+1})$, for which Properties R1 – R6 hold. Consider the distinguished set S_i^* of $\mathcal{S}_i$ that we denote for simplicity by S^*, and its corresponding set J^* of boundary vertices that we denote by J^*. Recall that, from Requirement R3, $V(G_i) = S^* \setminus J^*$. Consider now the cut (A_{i+1}, B_{i+1}) in G_i. We extend this cut to a partition (A', B') of S^* as follows. Initially, we set $(A', B') = (A_{i+1}, B_{i+1})$. Next, we consider the boundary vertices $v \in J^*$ one by one. Recall that, from Property R4, only regular edges may be incident to v in $G_0[S^*]$. Therefore, if $v \in L$, then it has no incoming edges in $G_0[S^*]$; in this case we add it to B'. Similarly, if $v \in R$, then it has no outgoing edges in $G_0[S^*]$; in this case we add it to A'. Once all vertices of J^* are processed, we obtain the partition (A', B') of S^*. Notice that $E_{G_0}(A', B') = E_{i+1}$, and all edges of $E_{G_0}(B', A')$ are regular edges. In order to obtain sequence $\mathcal{S}_{i+1}$, we replace the set S^* in $\mathcal{S}_i$ with the sets A', B', in this order. It is immediate to verify that Properties R1, R5 and R6 continue to hold for $\mathcal{S}_{i+1}$.

If cut (A_{i+1}, B_{i+1}) is a type-1 cut, that is, $w(A_{i+1}) > w(B_{i+1})$, then we designate A' as the distinguished set S_{i+1}^* of vertices for $\mathcal{S}_{i+1}$, and we let $J_{i+1} \cup (A' \cap J^*)$ be the corresponding set J_{i+1}^* of boundary vertices. Alternatively, if (A_{i+1}, B_{i+1}) is a type-2 cut, then we designate B' as the distinguished set S_{i+1}^* of vertices for $\mathcal{S}_{i+1}$, and we let $J_{i+1}' \cup (B' \cap J^*)$ be the corresponding set J_{i+1}^* of boundary vertices. It is immediate to verify that, with this definition, Properties R2–R4 continue to hold. The algorithm consists of $q \leq O(\hat{W})$ iterations, and each iteration can be executed in time $O(\hat{W})$. Therefore, the running time of the algorithm is $O(\hat{W}^2)$.

We are now ready to define the partition (S_1, S_2, S_3) of the vertices of G_0. Consider the final sequence $\mathcal{Y}_q = (Y_1, \ldots, Y_{q+1})$, and assume that its distinguished set is Y_j. We denote by J^* the set of the boundary vertices of Y_j. Initially, we set $S_1 = \bigcup_{j'=1}^{j-1} Y_{j'}$, $S_2 = Y_j$, and $S_3 = \bigcup_{j'=j+1}^{q} Y_{j'}$. Clearly, (S_1, S_2, S_3) is a partition of $V(G_0)$, and, from properties R5 and R6, for every pair $1 \leq i < i' \leq 3$ of indices, for every edge $e = (x, y)$ with $x \in S_i, y \in S_{i'}$, e must be a special edge that lies in E', while for every edge $e' = (x', y')$ with $x' \in S_{i'}, y' \in S_i$, edge e' is a regular edge. Additionally, from Property

R3, $S_2 = V(G_q) \cup J^*$. In our last step we will move every vertex of J^* to either S_1 or S_3, in order to ensure that $S_2 = V(G_q)$ holds.

For convenience, if $e = (x, y)$ is an edge of G_0, with $x \in S_i$ and $y \in S_{i'}$, then we say that e is a *left-to-right* edge if $i < i'$, and that it is a *right-to-left* edge if $i' < i$. Notice that the statement of the claim requires that all left-to-right edges are special edges that lie in E', and all right-to-left edges are regular edges.

We consider every vertex $v \in J^*$ one by one. Recall that, from Property R4, all edges incident to v in $G_0[Y_j]$ are regular edges. Assume first that $v \in L$. Let $e = (u, v)$ be the unique special edge incident to v. Then $u \in S_1$ must hold, and e is the only edge entering v. We move v from S_2 to S_3. Notice that, if e' is a regular edge incident to v, then $e' = (v, u')$ must hold for some vertex u', and so this move does not create any new left-to-right edges, while it may only introduce new right-to-left edges that are regualr edges.

Assume now that $v \in R$. Let $e = (v, u)$ be the unique special edge incident to v. Then we are guaranteed that $u \in S_3$ must hold. Edge e is the only edge that leaves v in G_0, and all other edges incident to v are incoming edges, which must also be regular edges. We move vertex v from S_2 to S_1. As before, this move does not introduce any new left-to-right edges, and the only new right-to-left edges that it may introduce are regular edges that are incident to v.

Once every vertex of J^* is processed we obtain the partition (S_1, S_2, S_3) with the required properties. It is easy to verify that the running time of the algorithm is $O((\hat{W})^2)$. $\square$

Note that our updates to the graph G after each cut (A_i, B_i) ensure that G remains a perfect well-structured graph throughout the algorithm, so in particular, for every vertex $v \in S_2 = V(G_q)$, if e is the unique special edge that is incident to v in G_0, then its other endpoint u also lies in S_2. The partition (S_1, S_2, S_3) of the vertices of G_0 naturally defines a partition $(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ of the vertices of H_0, as follows. First, for every vertex $v \in V(G_0)$, if $v \in S_i$, for $1 \le i \le 3$, then we add v to $\tilde{S}_i$ as well. Next, we consider the special edges of $E(G_0)$. Consider any such special edge $e = (x, y)$, and assume that $x \in S_i$ and $y \in S_{i'}$, for indices $1 \le i, i' \le 3$. If $i = i'$, then we add all vertices lying on the path $P^*(e)$ to set $\tilde{S}_i$. Otherwise, from our construction $i < i'$ must hold, and moreover, from the above discussion, $i = 1$ and $i' = 3$ must hold. We then add all inner vertices of $P^*(e)$ to set $\tilde{S}_1$. Consider the resulting partition $(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ of the vertices of $V(H_0)$. As before, if $e = (x, y)$ is any edge of H_0 with $x \in \tilde{S}_i$ and $y \in \tilde{S}_{i'}$, then we say that e is a *left-to-right* edge if $i < i'$ and we say that it is a *right-to-left* edge if $i' < i$. Our construction insures that, if e' is a left-to-right edge of H_0 with respect to the partition $(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$, then its parent-edge e is a left-to-right edge of G_0 with respect to the partition (S_1, S_2, S_3). Moreover, if e is a left-to-right edge of G_0 with respect to the partition (S_1, S_2, S_3), then exactly one edge of $P^*(e)$ is a left-to-right edge of H_0 with respect to the partition $(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$. We conclude that the total number of left-to-right edges in H_0 with respect to the partition $(\tilde{S}_1, \tilde{S}_2, \tilde{S}_3)$ is bounded by $|E'| \le \frac{8|T|}{\log^{30} N}$. Notice that, since the phase has not terminated so far, the total number of vertices that were deleted from H_0 since the beginning of the phase is at most $\frac{\hat{W}}{\log^{30} N}$, and so in particular, $|\tilde{S}_1| + |\tilde{S}_3| \le \frac{\hat{W}}{\log^{30} N}$.

Next, we consider two cases. The first case happens if $|\tilde{S}_2 \cap T| \le \left(1 - \frac{4}{\log^4 N}\right) \cdot |T|$. In this case, either $|\tilde{S}_1 \cap T| \ge \frac{2|T|}{\log^4 N}$, or $|\tilde{S}_3 \cap T| \ge \frac{2|T|}{\log^4 N}$. Assume first that the former holds. In this case, we return the cut $(\tilde{S}, \tilde{S}')$, where $\tilde{S} = \tilde{S}_1$ and $\tilde{S}' = \tilde{S}_2 \cup \tilde{S}_3$. From our discussion, $|E_{H_0}(\tilde{S}, \tilde{S}')| \le \frac{8|T|}{\log^{30} N} \le \frac{|T|}{\log^6 N}$. Since $|\tilde{S}_1| \le \frac{\hat{W}}{\log^{30} N}$ and $|\tilde{S}_1 \cap T| \ge \frac{2}{\log^4 N}$, we get that $\tilde{S}'$ contains all but at most $\frac{\hat{W}}{\log^{30} N} \le \frac{\hat{W}}{8}$ vertices of H_0, and at most $\left(1 - \frac{2}{\log^4 N}\right) |T|$ vertices of T.

Similarly, if $|\tilde{S}_3 \cap T| \geq \frac{2|T|}{\log^4 N}$, then we return the cut $(\tilde{S}, \tilde{S}')$, where $\tilde{S} = \tilde{S}_1 \cup \tilde{S}_2$ and $\tilde{S}' = \tilde{S}_3$. Using the same reasoning as before, it is easy to see that $\tilde{S}$ contains all but at most $\frac{\hat{W}}{\log^{30} N} \leq \frac{\hat{W}}{8}$ vertices of H_0, and at most $\left(1 - \frac{2}{\log^4 N}\right)|T|$ vertices of T.

It now remains to consider the second case, where $|\tilde{S}_2 \cap T| > \left(1 - \frac{4}{\log^4 N}\right) \cdot |T|$. Intuitively, we would now like to use the finalizing cut $(\tilde{A}, \tilde{B})$ in graph H_q, in order to compute the cut $(\tilde{S}, \tilde{S}')$. For example, we could start with $\tilde{S} = \tilde{S}_1$ and $\tilde{S}' = \tilde{S}_3$, and then add vertices of $\tilde{A}$ to $\tilde{S}$, and vertices of $\tilde{B}$ to $\tilde{S}'$. The technical issue with this approach is that $\tilde{S}_2$ is a subset of vertices of graph H_0. Specifically, $\tilde{S}_2$ is the set of vertices of the subdivided graph of $G_0[S_2]$. Intuitively, we can think of graph H_q as being obtained from $H_0[\tilde{S}_2]$ by iteratively subdividing some of the special edges. Therefore, not all vertices of H_q lie in $\tilde{S}_2$, and so it is not immediately clear how to use the cut $(\tilde{A}, \tilde{B})$ in order to obtain the cut $(\tilde{S}, \tilde{S}')$ in graph H_0 with the desired properties. In order to overcome this technical difficulty, we will slightly modify the cut $(\tilde{A}, \tilde{B})$ in H_q. The goal of this modification is to ensure that, for every special edge $e \in E(G_q)$, all inner vertices of $P^*(e)$ lie on the same side of the cut, and, if both endpoints of e lie on the same side of the cut, then so do all vertices of $P^*(e)$. At the same time we need to ensure that we preserve some of the useful properties of the cut $(\tilde{A}, \tilde{B})$: namely, that $|E_{H_q}(\tilde{A}, \tilde{B})|$ is small, and that one of the sides of the cut contains many vertices of H_q but relatively few vertices of T.

We provide the algorithm for modifying the cut $(\tilde{A}, \tilde{B})$ in the next observation. For simplicity, in the observation and its proof we denote the graph H_q by H and G_q by G.

Observation D.2 *There is a deterministic algorithm, that, given a finalizing cut $(\tilde{A}, \tilde{B})$ in graph H, computes a cut $(\tilde{A}', \tilde{B}')$ in H, such that the following hold:*

- *for each special edge $e = (x, y)$ of G, either all inner vertices on $P^*(e)$ lie in $\tilde{A}'$, or all inner vertices of $P^*(e)$ lie in $\tilde{B}'$. Moreover, if both endpoints of e lie in $\tilde{A}'$, then so do all vertices of $P^*(e)$, and if both endpoitns of e lie in $\tilde{B}'$, then so do all vertices of $P^*(e)$:*
- *all edges in $E_H(\tilde{A}', \tilde{B}')$ are special edges;*
- *$|E_H(\tilde{A}', \tilde{B}')| \leq \frac{|T|}{8 \log^6 N}$; and*
- *one of the sides of the cut $\tilde{Z} \in \left\{\tilde{A}', \tilde{B}'\right\}$ contains at least $|V(H)| - \frac{\hat{W}}{16}$ vertices of H, and at most $|T| \cdot \left(1 - \frac{6}{\log^4 N}\right)$ vertices of T.*

The running time of the algorithm is $O(\hat{W}^2)$.

We complete the proof of Observation D.2 below, after we complete the proof of Claim 7.9 using it. Consider some special edge $e \in E(G_q)$. For simplicity, we denote the subdivided path $P^*(e)$ in the corresponding graph H_q by $P'(e)$, while we use $P^*(e)$ to denote the subdivided path corresponding to e in H_0. Therefore, $P'(e)$ is obtained from $P^*(e)$ by iteratively subdividing some of the special edges.

We construct the cut $(\tilde{S}, \tilde{S}')$ in graph H_0 as follows. We start with $\tilde{S} = \tilde{S}_1$ and $\tilde{S}' = \tilde{S}_3$. Next, we consider all vertices $v \in S_2 = V(G_q)$. For each such vertex, if $v \in \tilde{A}'$, then we place v in $\tilde{S}$, and otherwise, we place it in $\tilde{S}'$. Lastly, we consider every special edge $e \in E(G_q)$. If all inner vertices of $P'(e)$ lie in $\tilde{A}'$, then we place all inner vertices of $P^*(e)$ in $\tilde{S}$; otherwise, we place all inner vertices of $P^*(e)$ in $\tilde{S}'$.

We now verify that the resulting cut $(\tilde{S}, \tilde{S})$ has all required properties. Consider first any edge $e \in E_{H_0}(\tilde{S}, \tilde{S}')$. If this edge is incident to a vertex of $\tilde{S}_1$ or to a vertex of $\tilde{S}_2$, then we are guaranteed that $e \in E'$ holds. Otherwise, there is a parent-edge $e' = (x, y) \in G_0$ of e, with $x \in \tilde{A}'$, $y \in \tilde{B}'$. In this case, path $P'(e')$ contributed exactly one special edge to the set $E_{H_q}(\tilde{A}', \tilde{B}')$ in H_q, and path $P^*(e')$ contributes exactly one special edge to the set $E_{H_0}(\tilde{S}, \tilde{S}')$ – the edge e. Therefore, every edge in $E_{H_0}(\tilde{S}, \tilde{S}')$ is a special edge, and:

$$|E_{H_0}(\tilde{S}, \tilde{S}')| \leq |E'| + |E_H(\tilde{A}', \tilde{B}')| \leq \frac{8|T|}{\log^{30} N} + \frac{|T|}{8 \log^6 N} \leq \frac{|T|}{4 \log^6 N}.$$

Recall that one of the sides of the cut $\tilde{Z} \in \left\{\tilde{A}', \tilde{B}'\right\}$ contains all but at most $\frac{\hat{W}}{16}$ vertices of H_q, and at most $|T| \cdot \left(1 - \frac{6}{\log^4 N}\right)$ vertices of T.

Assume first that $\tilde{Z} = \tilde{A}'$. Recall that we have established that $|\tilde{S}_3| \leq \frac{\hat{W}}{\log^{30} N}$. Since $|\tilde{B}'| \leq \frac{\hat{W}}{16}$, we get that $|\tilde{S}'| \leq |\tilde{S}_3| + |\tilde{B}'| \leq \frac{\hat{W}}{\log^{30} N} + \frac{\hat{W}}{16} \leq \frac{\hat{W}}{8}$. Additionally, since we have assumed that $|\tilde{S}_2 \cap T| > \left(1 - \frac{4}{\log^4 N}\right) \cdot |T|$, we get that $|\tilde{S}_1 \cap T| \leq \frac{4|T|}{\log^4 N}$, and so $|\tilde{S} \cap T| \leq |\tilde{S}_1 \cap T| + |\tilde{A}' \cap T| \leq \left(1 - \frac{2}{\log^4 N}\right) \cdot |T|$.

Similarly, if $\tilde{Z} = \tilde{B}'$, then $|\tilde{S}| \leq |\tilde{S}_1| + |\tilde{A}'| \leq \frac{\hat{W}}{\log^{30} N} + \frac{\hat{W}}{16} \leq \frac{\hat{W}}{8}$. Since we have assumed that $|\tilde{S}_2 \cap T| > \left(1 - \frac{4}{\log^4 N}\right) \cdot |T|$, we get that $|\tilde{S}_3 \cap T| \leq \frac{4|T|}{\log^4 N}$, and so $|\tilde{S}' \cap T| \leq |\tilde{S}_3 \cap T| + |\tilde{B}' \cap T| \leq \left(1 - \frac{2}{\log^4 N}\right) \cdot |T|$.

We now analyze the running time of the algorithm. The running times of the algorithms from Claim D.1 and from Observation D.2 are bounded by $O(\hat{W}^2)$. It is easy to verify that the running time of the remainder of the algorithm is bounded by $O(\hat{W}^2)$ as well, so that total running time of the algorithm is bounded by $O(\hat{W}^2)$.

In order to complete the proof of Claim 7.9, it is now enough to prove Observation D.2, which we do next. In the proof of the observation we refer to graph H_q as H, and to graph G_q as G.

Proof of Observation D.2. We partition the set of all special edges of $E(G)$ into four subsets. The first set, E^A, contains all special edges (x, y) with $x, y \in \tilde{A}$. The second set, E^B, contains all special edges (x, y) with $x, y \in \tilde{B}$. The third set, E^{AB} contains all special edges (x, y) with $x \in \tilde{A}$ and $y \in \tilde{B}$, and the fourth set, E^{BA}, contains all special edges (x, y) with $x \in \tilde{B}$ and $y \in \tilde{A}$. We say that a a special edge $e \in E(G)$ is *bad*, if some edge of $P^*(e)$ lies in $E_H(\tilde{A}, \tilde{B})$. Clearly, the number of bad special edges is bounded by $|E_H(\tilde{A}, \tilde{B})| \leq \frac{|T|}{8 \log^6 N}$. Recall that, since cut $(\tilde{A}, \tilde{B})$ is finalizing, one of the sides of the cut $Z \in \left\{\tilde{A}, \tilde{B}\right\}$ contains at least $|V(H)| - \frac{\hat{W}}{32}$ vertices of H, and at most $|T| \cdot \left(1 - \frac{7}{\log^4 N}\right)$ vertices of T.

We start with $(\tilde{A}', \tilde{B}') = (\tilde{A}, \tilde{B})$, and then modify the cut $(\tilde{A}', \tilde{B}')$, in two steps. In the first step, we consider every bad edge $e \in E(G)$ in turn. If $e \in E^A \cup E^{AB} \cup E^{BA}$, then we move all inner vertices of $P^*(e)$ to set $\tilde{A}'$. Otherwise, $e \in E^B$ must hold, and we move all inner vertices of $P^*(e)$ to $\tilde{B}'$.

Consider the resulting cut $(\tilde{A}', \tilde{B}')$, that is obtained after every bad special edge of G has been processed. It is immediate to verify that $|E_H(\tilde{A}', \tilde{B}')| \leq |E_H(\tilde{A}, \tilde{B})| \leq \frac{|T|}{8 \log^6 N}$ continues to hold. Indeed, every bad edge $e \in E(G)$ contributes at least one edge of $P^*(e)$ to $E_H(\tilde{A}, \tilde{B})$, and it contributes at most one edge of $P^*(e)$ to $E_H(\tilde{A}', \tilde{B}')$. It is immediate to verify that every edge in $E_H(\tilde{A}', \tilde{B}')$ is a special edge, since, for every special edge e of G, the first and the last edge on path $P^*(e)$ are special edges of H. Moreover, every edge in $E_H(\tilde{A}', \tilde{B}')$ is a special edge that is a child-edge of some bad edge in E^{AB}.

Recall that the number of bad edges in G is bounded by $|E_H(\tilde{A}, \tilde{B})| \leq \frac{|T|}{8 \log^6 N}$. Consider the set $Z \in \left\{\tilde{A}, \tilde{B}\right\}$, such that contains at least $|V(H)| - \frac{\hat{W}}{32}$ vertices of H, and at most $|T| \cdot \left(1 - \frac{7}{\log^4 N}\right)$

vertices of T. Since Event $\mathcal{E}_4$ did not happen, for every bad edge $e \in E(G)$, the correpsonding path $P^*(e)$ contains at most $16 \log N$ vertices of T. Therefore, the total number of vertices of T that our procedure could have added to Z during the first step is bounded by $\frac{|T|}{8 \log^6 N} \cdot 16 \log N \leq \frac{2|T|}{\log^5 N}$. Therefore, after the transformation, Z contains at most $|T| \cdot \left(1 - \frac{6}{\log^4 N}\right)$ vertices of T. Additionally, since the length of every edge e in G is bounded by $d'/2$, for every special edge $e \in E(G)$, the number of vertices on path $P^*(e)$ is bounded by d'. Therefore, the total number of vertices that our algorithm may have removed from Z is bounded by $d' \cdot \frac{|T|}{8 \log^6 N} \leq \frac{\hat{W}}{\log^6 N}$ (since $|T| \leq \frac{8\hat{W}}{d'}$, as Event $\mathcal{E}_1$ did not happen). Therefore, after the first step, Z still contains at least $|V(H)| - \frac{3\hat{W}}{64}$ vertices. To summarize, after the first step, Z contains at least $|V(H)| - \frac{3\hat{W}}{64}$ vertices, and at most $|T| \cdot \left(1 - \frac{6}{\log^4 N}\right)$ vertices of T.

Consider any special edge $e = (x, y)$ of G. If $x, y \in \tilde{A}'$, and any vertex of $P^*(e)$ originally lied in $\tilde{B}'$, then e must have been a bad edge, as some edge on $P^*(e)$ must have lied in $E_H(\tilde{A}, \tilde{B})$. Our algorithm then moved all inner vertices on $P^*(e)$ to $\tilde{A}'$. Simiarly, if $x, y \in \tilde{B}'$, then our algorithm ensures that all inner vertices on $P^*(e)$ lie in $\tilde{B}'$. If $x \in \tilde{A}$ and $y \in \tilde{B}$ held, then e must be a bad edge, since some edge on $P^*(e)$ must lie in $E_H(\tilde{A}, \tilde{B})$. Our algorithm moved all inner vertices of $P^*(e)$ to $\tilde{A}'$. Lastly, assume that $x \in \tilde{B}'$ and $y \in \tilde{A}'$ holds. If e is a bad edge, then we have moved all inner vertices of $P^*(e)$ to $\tilde{A}'$. It now only remains to take care of the subset $E^* \subseteq E^{BA}$ of special edges, that contains every edge $e \in E^{BA}$ that is not bad. We take care of these edges in the second step.

Intuitively, in the second step, we consider each edge $e \in E^*$ one by one, and for each such edge, we move all inner vertices of $P^*(e)$ to one side of the cut. However, we need to be careful so that we do not remove too many vertices from the set Z, and that we do not add too many vertices of T to Z.

We define a subset $E_1^* \subseteq E^*$ of edges to contain every edge $e \in E^*$, such that the path $P^*(e)$ contains at least one vertex of T. If $|E_1^*| > \left\lceil \frac{6|T|}{\log^4 N} \right\rceil$, then we discard edges arbitrarily from E_1^*, until $|E_1^*| = \left\lceil \frac{6|T|}{\log^4 N} \right\rceil$ holds.

We now consider two cases. The first case happens when $Z = \tilde{A}$. In this case, for every edge $e \in E_1^*$, we move all inner vertices on path $P^*(e)$ to $\tilde{B}'$, and for every edge $e \in E^* \setminus E_1^*$, we move all inner vertices on path $P^*(e)$ to $\tilde{A}'$. Notice that, if $|E_1^*| = \frac{6|T|}{\log^4 N}$, then this process guarantees that $|\tilde{B}' \cap T| \geq \frac{6|T|}{\log^4 N}$ holds at the end of the second step, and so $|\tilde{A}' \cap T| \leq |T| \cdot \left(1 - \frac{6}{\log^4 N}\right)$ holds. If $|E_1^*| < \frac{6|T|}{\log^4 N}$, then we did not add any new terminals to set $\tilde{A}'$ in this step, and so $|\tilde{A}' \cap T| \leq |T| \cdot \left(1 - \frac{6}{\log^4 N}\right)$ continues to hold. Since, for every edge $e \in E_1^*$, the number of inner vertices on path $P^*(e)$ is bounded by d', the number of vertices that we have deleted from $\tilde{A}'$ in this step is bounded by:

$$d' \cdot |E_1^*| \leq d' \cdot \left\lceil \frac{6|T|}{\log^4 N} \right\rceil \leq d' \cdot \frac{100\hat{W}}{d' \log^4 N} \leq \frac{\hat{W}}{\log^3 N}.$$

(We have used the fact that $|T| \leq \frac{8\hat{W}}{d'}$, since Event $\mathcal{E}_1$ did not happen). Therefore, after the second step, $\tilde{A}'$ contains at least $|V(H)| - \frac{3\hat{W}}{64} - \frac{\hat{W}}{\log^3 N} \geq |V(H)| - \frac{\hat{W}}{16}$ vertices.

If $Z = \tilde{B}$, then for every edge $e \in E_1^*$, we move all inner vertices on path $P^*(e)$ to $\tilde{A}'$, and for every edge $e \in E^* \setminus E_1^*$, we move all inner vertices on path $P^*(e)$ to $\tilde{B}'$. Using the same arguments as before, we conclude that, after the second step, $|\tilde{B}' \cap T| \leq |T| \cdot \left(1 - \frac{6}{\log^4 N}\right)$ and $|\tilde{B}'| \geq |V(H)| - \frac{\hat{W}}{8}$ hold.

It is easy to verify that the time required to compute the modified cut $(\tilde{A}', \tilde{B}')$ is bounded by

$O(|E(H)|) \le O(\hat{W}^2)$. □

D.3 Proof of Observation 7.14

It is enough to prove that inequalities $r \le \lceil\sqrt{\log N'}\rceil$, $1 \le \eta' \le \hat{\Delta}$, and $\frac{(|V(H)|-|B|)\cdot\eta'}{\hat{\Delta}} \le 2^{r\sqrt{\log N'}}$ all hold.

First, since $r \le \lceil\sqrt{\log N}\rceil$, and $N \le N'$, it is immediate to verify that $r \le \lceil\sqrt{\log N'}\rceil$.

Second, since Event $\mathcal{E}_1$ did not happen, $\hat{\Delta} \ge \frac{|T|}{4} \ge \frac{\hat{W}}{8d'}$. Since $\eta' = (\log N')^8$, we then get that:

$$\hat{\Delta} \ge \frac{\hat{W}}{8d'} \ge \frac{n}{16d'} \ge \frac{n\cdot(\log N)^{28}}{32d} \ge \frac{(\log N)^{28}}{64} \ge \eta'.$$

(we have used the fact that $\hat{W} \ge |V(G)| \ge \frac{n}{2}$; $d' \le \frac{2d}{(\log N)^{28}}$; $d \le n$ from Inequaltiy 15), and $\eta' = (\log N')^8$. Therefore, $1 \le \eta' \le \hat{\Delta}$ holds.

Lastly:

$$\frac{(|V(H)|-|B|)\cdot\eta'}{\hat{\Delta}} \le \frac{\hat{W}\cdot\eta'}{\hat{\Delta}} \le 8d'(\log N')^8 \le d \le 2^{r\cdot\sqrt{\log N}} \le 2^{r\cdot\sqrt{\log N'}}.$$

D.4 Proof of Claim 7.15

We start by turning the cut (Z, Z') into a weakly well-structured one, using the following straightforward algorithm. Consider every vertex $v \in Z$ one by one. If there is a regular edge $(v,u) \in E_H(Z,Z')$, then we move v to Z'. Notice that, if a regular edge (v,u) exists, then $v \in L$ must hold, so moving v to Z' may introduce at most one new edge into the set $E_H(Z,Z')$ – the special edge that is incident to v. At the same time, at least one regular edge is deleted from $E_H(Z,Z')$, so $|E_H(Z,Z')|$ does not increase. At the end of this procedure, cut (Z,Z') becomes weakly well-structured, and $|E_H(Z,Z')| \le \frac{\hat{W}}{d'(\log N)^7}$ continues to hold. Since $|T| \ge \frac{\hat{W}}{2d'}$, as Event $\mathcal{E}_1$ did not happen, we get that $|E_H(Z,Z')| \le \frac{2|T|}{\log^7 N}$. The number of vertices that were deleted from Z is bounded by the original number of edges in $E_H(Z,Z')$, which is bounded by $\frac{\hat{W}}{d'(\log N)^7} \le \frac{2|T|}{\log^7 N} \le \frac{|T|}{8\log^6 N}$. Therefore, at the end of this procedure, $|Z \cap T|, |Z' \cap T| \ge \frac{7|T|}{8\log^4 N}$ holds. Notice that the running time of the algorithm so far is bounded by $O(|E(H)|) \le O(\hat{W}^2)$. If either of the sets Z or Z' contains at least $|V(H)| - \frac{\hat{W}}{32} = \frac{31\hat{W}}{32}$ vertices, then cut (Z,Z') is a finalizing cut (see Definition 7.7). We then terminate the algorithm and return this cut.

We assume from now on that $|Z|, |Z'| \ge \frac{\hat{W}}{32}$. Since $|E_H(Z,Z')| \le \frac{\hat{W}}{d'(\log N)^7} \le \frac{32}{d'\cdot(\log N)^7}\cdot\min\{|Z|,|Z'|\}$, we get that the sparsity of the cut (Z,Z') is at most $\frac{32}{d'\cdot(\log N)^7}$.

Next, we turn the cut (Z,Z') into a cut (A',B') in G, in a natural way: we let $A' = Z \cap V(G)$, and $B' = Z' \cap V(G)$. Recall that the length of every edge in G is at most $\frac{d'}{2}$. From Observation 4.4, (A',B') is a weakly well-structured cut in G of sparsity at most $\frac{256}{d'\log^7 N}$, with $w(A') \ge \frac{Z}{8} \ge \frac{\hat{W}}{256}$ and $w(B') \ge \frac{|Z'|}{8} \ge \frac{\hat{W}}{256}$. The time required to compute the cut (A',B') in G given the cut (Z,Z') in H is bounded by $O(|V(H)|) \le O(\hat{W})$.

Recall that for every special edge $e \in E(G)$, $\ell(e) \leq \frac{d'}{2} \leq \frac{n}{16}$, while $\sum_{e\in E(G)} \ell(e) \geq \frac{n}{4}$. We can now apply the algorithm from Observation 4.5 to cut (A', B'), to compute a strongly well-structured cut (A, B) in G with $w(A) \geq \frac{w(A')}{4} \geq \frac{\hat{W}}{1024} \geq \frac{\hat{W}}{\log^{30} N}$ and $w(B) \geq \frac{w(B')}{4} \geq \frac{\hat{W}}{\log^{30} N}$, whose sparsity is at most $\frac{1024}{d' \log^7 N} \leq \frac{1}{d'}$. The running time of the algorithm from Observation 4.5 is $O(|E(H)|) \leq O(\hat{W}^2)$. We then return the cut (A, B).

From the above discussion, the total running time of the algorithm is $O(\hat{W})^2$.

D.5 Proof of Claim 7.21

Clearly, G is a perfect well-structured graph, and the assignment $\ell'(e)$ of lengths to its edges $e \in E(G)$ is a proper assignment. The current weight of all vertices in G with respect to the edge lengths $\ell'(\cdot)$ is $W' \leq 2\hat{W} \leq 24W^0$, while $N \geq W^0$ holds from the definition of the MaintainCluster problem. Therefore, $N' = 8N \geq 8W^0 \geq 8n \geq 8|V(G)|$, and $N' = 8N \geq 8W^0 \geq \frac{W'}{8}$ holds, so $N' \geq \max\left\{|V(G)|, \frac{W'}{8}\right\}$, as required. Since N is large enough from the definition of the MaintainCluster problem, so is N'. Since Event $\mathcal{E}_1$ did not happen, $|\beta| = |V(X)| = |T| \geq \frac{\hat{W}}{2d'} \geq \frac{W'}{256d'}$, as required. Since $1 \leq r \leq \lceil\sqrt{\log N}\rceil$ holds from the definition of r-restricted MaintainCluster problem, and $N' \geq N$, we get that $1 \leq r \leq \lceil\sqrt{\log N'}\rceil$. Lastly, we need to verify that $d' \leq \min\left\{\frac{2^{r\sqrt{\log N'}}}{\log^8 N'}, \frac{|V(G)|}{8}\right\}$ holds. Indeed, recall that $d' \leq \frac{2d}{(\log N)^{28}}$. Moreover, from Inequality 15, $d \leq n$, and from the definition of the r-restricted MaintainCluster problem, $d \leq 2^{r\cdot\sqrt{\log N}}$. Therefore:

$$\begin{aligned}
d' &\leq \frac{2d}{(\log N)^{28}}\\
&\leq \min\left\{\frac{2n}{(\log N)^{28}}, \frac{2\cdot 2^{r\cdot\sqrt{\log N}}}{(\log N)^{28}}\right\}\\
&\leq \min\left\{\frac{2^{30}\cdot n}{(\log N')^{28}}, \frac{2^{29}\cdot 2^{r\cdot\sqrt{\log N'}}}{(\log N')^{28}}\right\}\\
&\leq \min\left\{\frac{|V(G)|}{8}, \frac{2^{r\sqrt{\log N'}}}{\log^8 N'}\right\}.
\end{aligned}$$

(We have used the fact that, from Inequality 17, $\log N' \leq 2\log N$).

References

[AAP93] Baruch Awerbuch, Yossi Azar, and Serge A. Plotkin. Throughput-competitive on-line routing. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 32–40, 1993.

[ABF23] Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 391–404. ACM, 2023.

[ABKZ22] Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in p via short cycle removal: Cycle detection, distance oracles, and beyond. *arXiv preprint arXiv:2204.10465*, 2022.

[AMV20] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 93–104, 2020.

[Ber17] Aaron Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 80. Schloss Dagstuhl-Leibniz-Center for Computer Science, 2017.

[BGS20] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1123–1134. IEEE, 2020.

[BGWN20] Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Near-optimal decremental sssp in dense weighted digraphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1112–1122. IEEE, 2020.

[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.

[CK24] Julia Chuzhoy and Sanjeev Khanna. A faster combinatorial algorithm for maximum bipartite matching. In *Proceedings of the Thirty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2024.

[CKL$^+$22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623, 2022.

[CMSV17] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in õ $(m^{10/7} \log W)$ time (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 752–771, 2017.

[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.

[DP09] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms.* Cambridge University Press, 2009.

[DS08] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In Cynthia Dwork, editor, *Proceedings of the 40th Annual*

ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008, pages 451–460. ACM, 2008.

[ES81] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

[FF56] L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[Fle00] Lisa Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

[Gab17] Harold N. Gabow. The weighted matching approach to maximum cardinality matching. *Fundam. Informaticae*, 154(1-4):109–130, 2017.

[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.

[GK04] Andrew V. Goldberg and Alexander V. Karzanov. Maximum skew-symmetric flows and matchings. *Math. Program.*, 100(3):537–568, 2004.

[GKK10] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Perfect matchings in o(n log n) time in regular bipartite graphs. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 39–46, 2010.

[GWN20] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Decremental sssp in weighted digraphs: Faster and against an adaptive adversary. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2542–2561. SIAM, 2020.

[HK73] John E. Hopcroft and Richard M. Karp. An $\mathrm{n}^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[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.

[IM81] Oscar H. Ibarra and Shlomo Moran. Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication. *Inf. Process. Lett.*, 13(1):12–15, 1981.

[Kin99] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 81–89. IEEE, 1999.

[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.

[KRV09] Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)*, 56(4):19, 2009.

[Lou10] Anand Louis. Cut-matching games on directed graphs. *CoRR*, abs/1010.1047, 2010.

[LRS13] Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 755–764, 2013.

[LS19] Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt(rank) linear system solves. *CoRR*, abs/1910.08033, 2019.

[LS20] Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 803–814, 2020.

[Mad10] Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 121–130, 2010.

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 253–262, 2013.

[Mad16] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 593–602, 2016.

[Mar73] G. A. Margulis. Explicit construction of concentrators. *Problemy Peredafi Iqfiwmacii*, 9(4):71–80, 1973. (English translation in Problems Inform. Transmission (1975)).

[MS04] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 248–255, 2004.

[MV80] Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980.

[RZ11] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011.

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, 2003.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90, 2004.

[Vaz94] Vijay V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\text{sqrt}\{v\ e\})$ general graph maximum matching algorithm. *Comb.*, 14(1):71–109, 1994.

[vdBCK$^+$23] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th IEEE Annual Symposium on Foundations of Computer Science (FOCS), (to appear)*, 2023.

[vdBLN$^+$20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 919–930, 2020.