Breaking the $O(mn)$-Time Barrier for Vertex-Weighted Global Minimum Cut*

Julia Chuzhoy† Ohad Trabelsi‡

Abstract

We consider the Global Minimum Vertex-Cut problem: given an undirected vertex-weighted graph G, compute a minimum-weight subset of its vertices whose removal disconnects G. The problem is closely related to Global Minimum Edge-Cut, where the weights are on the graph edges instead of vertices, and the goal is to compute a minimum-weight subset of edges whose removal disconnects the graph. Global Minimum Cut is one of the most basic and extensively studied problems in combinatorial optimization and graph theory. While an almost-linear time algorithm was known for the edge version of the problem for awhile (Karger, STOC 1996 and J. ACM 2000), the fastest previous algorithm for the vertex version (Henzinger, Rao and Gabow, FOCS 1996 and J. Algorithms 2000) achieves a running time of $\tilde{O}(mn)$, where m and n denote the number of edges and vertices in the input graph, respectively. For the special case of unit vertex weights, this bound was broken only recently (Li et al., STOC 2021); their result, combined with the recent breakthrough almost-linear time algorithm for Maximum s-t Flow (Chen et al., FOCS 2022, van den Brand et al., FOCS 2023), yields an almost-linear time algorithm for Global Minimum Vertex-Cut with unit vertex weights.

In this paper we break the 28 years old bound of Henzinger et al. for the general weighted Global Minimum Vertex-Cut, by providing a randomized algorithm for the problem with running time $O(\min\{mn^{0.99+o(1)}, m^{1.5+o(1)}\})$.

*STOC 2025, to appear

†Toyota Technological Institute at Chicago. Email: cjulia@ttic.edu. Supported in part by NSF grant CCF-2402283 and NSF HDR TRIPODS award 2216899.

‡Toyota Technological Institute at Chicago. Email: ohadt@ttic.edu.

Contents

1 Introduction

We consider the Global Minimum Vertex-Cut problem: given an undirected n-vertex and m-edge graph $G = (V, E)$ with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V$, compute a subset $S \subseteq V$ of vertices, such that the graph $G \setminus S$ is not connected. A closely related and extensively studied problem is Global Minimum Edge-Cut: here, the input graph has weights on its edges instead of vertices, and the goal is to compute a minimum-weight subset of edges, whose removal disconnects the graph. The Global Minimum Cut problem is among the most fundamental and extensively studied in computer science, with algorithms dating back to the 1960's [Kle69, Pod73, ET75, LLW88]. The famous edge-contraction based algorithm of Karger and Stein [KS93] for the edge version of the problem is one of the gems of Theoretical Computer Science and is routinely taught in basic Algorithms classes. The notion of a Global Minimum Cut is closely related to the central Graph-Theoretic notions of edge- and vertex-connectivity. The edge-connectivity of a graph G is the smallest value λ, such that, for every pair u, v of vertices of G, there is a collection λ of edge-disjoint paths connecting u to v. Vertex-connectivity is defined similarly, except that the paths are required to be internally vertex-disjoint. By Menger's Theorem [Men27], if G is a graph with unit vertex weights, then the value of the Global Minimum Vertex-Cut in G is equal to its vertex-connectivity. One can naturally extend the notion of vertex-connectivity to the vertex-weighted setting, where a collection of λ internally disjoint u-v paths is replaced with a u-v flow of value λ, that obeys the vertex capacities defined by their weights. Similarly, the value of the Global Minimum Edge-Cut in a graph with unit edge weights is equal to its edge-connectivity, and the value of the Global Minimum Edge-Cut in a graph with arbitrary edge weights can be thought as corresponding to the "weighted" notion of edge-connectivity, where edge-disjoint paths are replaced by flows.

The edge version of the Global Minimum Cut problem has been studied extensively, and is now reasonably well understood in undirected graphs. The classical algorithm of Karger and Stein [KS93] mentioned above achieves running time $\tilde{O}(n^2)$, improving upon the previous fastest $\tilde{O}(mn)$-time algorithm by Hao and Orlin [HO94]. Later, Karger [Kar00] obtained a randomized algorithm with near optimal running time $\tilde{O}(m)$, using tree-packing and dynamic programming. This running time was recently matched by a deterministic algorithm [HLRW24]. The same problem was also studied in the directed setting: Cen et al. [CLN$^+$21], building on some of the ideas from [Kar00], recently presented a randomized algorithm for the directed edge version with running time $O(\min\{n/m^{1/3}, \sqrt{n}\} \cdot m^{1+o(1)})$.

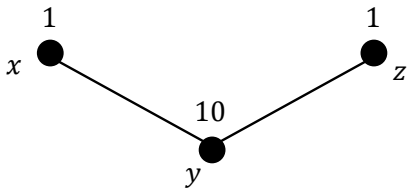
Despite this, progress on the vertex-cut version of the problem has been much slower. In 1996, Henzinger et al. [HRG00] presented an algorithm for vertex-weighted Global Minimum Cut with running time $\tilde{O}(mn)$. In the 28 years since then, no significant progress has been made on the general vertex-weighted version. This lack of progress is likely due to several existing technical barriers to adapting the approaches used for the edge-cut version of the problem. For example, while there are at most $\binom{n}{2}$ minimum cuts in the undirected edge-cut setting, the number of minimum vertex-cuts can be as large as $\Theta(2^k \cdot (n/k)^2)$ even in the unweighted setting, where k is the value of the optimal solution [Kan90]. Furthermore, the tree-packing method, such as the one used in [Kar00, CLN$^+$21], is not known to provide similar guarantees in the vertex capacitated setting. Only recently, the $O(mn)$ barrier was broken for the *unweighted* version of Global Minimum Vertex-Cut by Li et al. [LNP$^+$21]. Their algorithm, combined with the recent breakthrough almost-linear time algorithm for Maximum s-t Flow [CKL$^+$22, vdBCP$^+$23], achieves a running time of $O(m^{1+o(1)})$. They also obtain an $O(\min\{\tilde{O}(n^2), mn^{1-1/12+o(1)}\})$-time algorithm for the directed version of the problem with unit vertex weights.

We note that the recent progress on fast algorithms for Maximum s-t Flow mentioned above does not appear to directly imply algorithms for Global Minimum Cut with general vertex weights whose running time is below $O(mn)$. Moreover, while, in the context of Maximum s-t Flow, it is well known

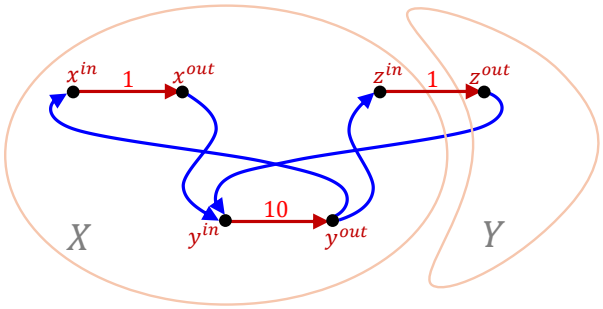
that the vertex-capacitated version of the problem in undirected graphs can be cast as a special case of the directed edge-capacitated version, this connection is not known in the context of Global Minimum Vertex-Cut. Unfortunately, the standard reduction, where an undirected vertex-weighted graph is replaced by the corresponding edge-weighted directed *split graph*[1], does not work for the Global Minimum Vertex-Cut problem (see Figure 1), and so the recent algorithms for the directed Global Minimum Edge-Cut problem of [CLN+21] cannot be directly leveraged to obtain a faster algorithm for undirected Global Minimum Vertex-Cut. This raises the following fundamental question:

Q1: *Can the Global Minimum Vertex-Cut problem be solved in time faster than* $\Theta(mn)$*?*

In this paper, we answer this question affirmatively, by presenting a randomized algorithm for weighted Global Minimum Vertex-Cut with running time $O(\min\{mn^{0.99+o(1)}, m^{3/2+o(1)}\})$. Our result also resolves an open question posed by [NSY19] regarding the existence of an $o(n^2)$-time algorithm for the problem, for the setting where $m = O(n)$. We introduce several new technical tools and expand the scope of some existing techniques; we hope that these tools and techniques will lead to even faster algorithms for the problem, and will be useful in other graph-based problems. We now provide a more detailed overview of previous work, followed by the formal statement of our results and a high-level overview of our techniques.



(a) A vertex-weighted graph G. The unique global minimum vertex-cut in G is $(\{x\}, \{y\}, \{z\})$, of value $\mathsf{OPT} = 10$.



(b) The corresponding split graph G'; the blue edges have infinite capacity. Consider the cut (X, Y), where $Y = \{z^{\text{out}}\}$ and $X = V(G')\setminus Y$. The cut value is 1, but it does not define a vertex cut in G of the same value.

Figure 1: An illustration of the classic split-graph reduction from undirected vertex-weighted graphs to directed edge-weighted graphs, in the context of Global Minimum Vertex-Cut.

1.1 Overview of Previous Related Work

In this subsection we provide a high-level overview of previous results for central variants of the Global Minimum Cut problem, and highlight the main challenges in adapting the techniques used in these results to the weighted Vertex-Cut setting.

Undirected edge-cut version. The famous algorithm of Karger and Stein [KS93] mentioned above achieves an $\tilde{O}(n^2)$ running time for the problem. The algorithm is iterative. In every iteration, an edge (u, v) is sampled with a probability proportional to its weight, and then u and v are contracted

[1] A split graph of a vertex-weighted undirected graph G is obtained by replacing every vertex $v \in V(G)$ by a directed edge $(v^{\text{in}}, v^{\text{out}})$ of capacity $w(v)$, and every edge (x, y) of G by a pair of edges $(x^{\text{out}}, y^{\text{in}})$ and $(y^{\text{out}}, x^{\text{in}})$ of infinite capacity.

into a single vertex. This process continues until only two vertices remain in the graph, that naturally define an edge-cut, which is then returned by the algorithm. Intuitively, the algorithm succeeds in computing a Global Minimum Edge-Cut with a reasonably high probability, because, if we fix some optimal cut (X, Y), then the edges crossing the cut have a relatively small weight, making it not very likely that an edge crossing the cut is chosen for contraction. This method is clearly tailored to the (undirected) edge-cut version of the problem.

In a later paper, Karger [Kar00] presented an $\tilde{O}(m)$-time algorithm, that achives a near-optimal running time for the same setting. This algorithm is based on the seminal result by Tutte and Nash-Williams [Tut61, NW61], which showed that any unweighted c-edge-connected graph contains a collection of $c/2$ edge-disjoint spanning trees. Karger [Kar00] made a crucial observation that the edges of a minimum cut must be partitioned across these $c/2$ spanning trees, and so there must be some tree T in the packing, such that the number of edges of T crossing the cut is at most 2; if the number of edges of a tree T crossing a cut (X, Y) is at most i, then we say that the cut *i-respects* the tree. To find the global minimum cut, the algorithm identifies, for each tree in the packing, the minimum cut among those that 2-respect the tree, and returns the smallest resulting cut. To make this approach efficient, the following strategy was used. First, in a preprocessing step, the input graph is sparsified by randomly subsampling edges in proportion to their capacities, and treating the resulting "skeleton" graph H as unweighted. This reduces the number of graph edges to $\tilde{O}(n)$ and the minimum cut value to $O(\log n)$, while ensuring that, if (X, Y) is a minimum cut in G, then its value in H is close to the value of the optimal cut in H. Next, $O(\log n)$ trees are packed into H using the algorithm of Gabow [Gab95]. Finally, dynamic programming is used to examine all trees in $\tilde{O}(m)$ time, in order to compute a minimum cut in the original graph. A sequence of works aimed at achieving fast deterministic algorithms for undirected Global Minimum Edge-Cut [HRW17, KT19, Li21] recently culminated with an $\tilde{O}(m)$-time algorithm [HLRW24].

Directed edge-cut version. The fastest current algorithm for this version, due to [CLN$^+$21], solves the problem in the time of $O(\min\{n/m^{1/3}, \sqrt{n}\})$ applications of Maximum s-t Flow, that, combined with the recent almost-linear time algorithm for the latter problem by [CKL$^+$22, vdBCP$^+$23], yields an $O(\min\{n/m^{1/3}, \sqrt{n}\} \cdot m^{1+o(1)})$-time algorithm for directed edge-weighted Global Minimum Cut. At a high level, the algorithm uses the ideas from [Kar00] described above. However, one significant barrier in this setting is that, in the context of directed graphs, a sparsifier that (approximately) preserves all cuts may not exist (see, e.g., [CCPS21]). In order to overcome this difficulty, the authors only rely on this method if the global minimum cut is unbalanced; since the number of unbalanced cuts can be suitably bounded even in directed graphs, arguments similar to those from [Kar00] can be used. The case of a balanced global minimum cut is handled by sampling vertices and computing Maximum s-t Flow between a subset of pairs of vertices from that sampled set; it is then shown that, with high probability, this subset contains at least one pair of vertices from opposite sides of the cut.

We note that the tree-packing based techniques used in the above algorithms appear to be unhelpful in the vertex-cut setting, as tree-packings have no known analogues in the vertex-connectivity regime that achieve similar guarantees (see [CGK14]).

Unweighted vertex-cut version. This version of the problem received a considerable amount of attention over the years [Kle69, Pod73, ET75, LLW88]. Recently, a series of results [NSY19, FNY$^+$20] culminated in an almost linear $O(m^{1+o(1)})$-time algorithm for this setting in undirected graphs, along with a somewhat similar $O(\min\{\tilde{O}(n^2), mn^{1-1/12+o(1)}\})$-time algorithm for directed graphs [LNP$^+$21]; both of the above bounds follow from the recent breakthrough almost-linear time algorithm for directed Maximum s-t Flow of [CKL$^+$22, vdBCP$^+$23]. Our algorithm for weighted Global Minimum Vertex-

Cut builds on several high-level ideas of [LNP+21], but requires a significant additional amount of work and new ideas in order to handle the weighted version of the problem. We provide a more detailed description of the algorithm of [LNP+21], the barriers to adapting it to the weighted version of the problem, and describe how our algorithm overcomes them in Section 1.2.

Weighted vertex-cut version. Henzinger et al. [HRG00] presented an algorithm with running time $\tilde{O}(mn)$ for this version of the problem, by generalizing the algorithm of Hao and Orlin [HO94] for the edge-cut version. At a high level, the algorithm of [HO94] identifies $n-1$ instances of the Maximum s-t Flow problem, such that one of the instances is guaranteed to correspond to the global minimum cut. However, instead of performing $n-1$ applications of the algorithm for Maximum s-t Flow, which would be too time consuming, Hao and Orlin showed that one can solve all these $n-1$ instances in time that is amortized to match the time of a single Maximum s-t Flow computation. However, this approach only works provided that the Maximum s-t Flow algorithm follows the Push-Relabel scheme [GT88]. Unfortunately, the best current such algorithm has running time $O(mn)$, and the recent almost linear-time algorithms for maximum s-t flow [CKL+22, vdBCP+23] do not follow the Push-Relabel paradigm and hence cannot be leveraged. Henzinger et al. [HRG00] refined this method of [HO94] and adapted it to the vertex-weighted version of the problem. Their algorithm uses the ideas from [BDD+82] to identify a collection of $\tilde{O}(n)$ instances of Maximum s-t Flow, whose solution is sufficient in order to compute global minimum vertex-cut. A combination of these two approaches then yields their $\tilde{O}(mn)$-time algorithm for Global Minimum Vertex-Cut. Another relevant work in this context is the recent combinatorial $O\left(n^{2+o(1)}\right)$-time algorithm for Maximum s-t Flow of [BBST24]. The algorithm proceeds by setting the initial s-t flow f to 0, and then iteratively augmenting it. In every iteration, an approximate Maximum s-t Flow f' in the residual flow network G_f is computed, via an algorithm that can be viewed as following the Push-Relabel scheme. The flow f is then augmented via this flow f', and the algorithm continues until an optimal flow f is obtained. Since this adaptation of the Push-Relabel scheme only computes an approximate flow in the residual flow network, to the best of our understanding, it cannot be directly used to obtain faster algorithms for Global Minimum Vertex-Cut using the approach of [HO94, HRG00] described above. Other recent developments on vertex-weighted Global Minimum Cut include a $(1+\varepsilon)$-approximation algorithm with running time $\tilde{O}(n^2/\varepsilon^2)$ by [CLN+21], and an exact algorithm with pseudo-polynomial running time $\tilde{O}(\mathsf{OPT} \cdot n \cdot \sum_{u \in V} w(u))$ by [CQ21].

Other Related Work. As mentioned already, there is a long and extensive line of work on Global Minimum Cut and its many variations. We summarize here some additional previous results that are most relevant to us, namely algorithms for the unweighted version of Global Minimum Vertex-Cut. We denote by k the value of the optimal solution (that may be as high as $\Omega(n)$), and by $T(m)$ the running time of the Maximum s-t Flow algorithm on an m-edge graph. One of the earliest algorithms for this problem by [Kle69] achieved a running time of $O(nk \cdot T(m))$ for undirected graphs. Subsequently, [Pod73, ET75] developed algorithms for directed graphs, with a similar running time $O(nk \cdot T(m))$. Even [Eve75] provided an algorithm with a running time of $(n+k^2) \cdot T(m)$, and later [BDD+82] designed an algorithm with a running time of $\tilde{O}(n) \cdot T(m)$ for undirected graphs. Linial, Lovász, and Wigderson [LLW88] discovered a surprising connection between the problem and convex hulls, that led to an algorithm with running time $\tilde{O}(n^\omega + nk^\omega)$ for undirected graphs, where ω is the matrix multiplication exponent, that simplifies to $\tilde{O}(n^{1+\omega})$ in the worst case when $k = \Theta(n)$. Their proof mimics a physical process, where an embedding of a graph is computed by viewing its edges as ideal springs, and then letting its vertices settle into an equilibrium. In a follow-up work, [CR94] extended their result to directed graphs. An algorithm with running time $O(k^3 n^{1.5} + k^2 n^2)$ was designed for the undirected version of the problem by [NI92, CKT93]. Finally, [Gab06] provided an

algorithm with running time $\tilde{O}(n + k\sqrt{n}) \cdot T(m) = \tilde{O}(n^{1.5}) \cdot T(m)$ by using expanders in order to identify a small number of vertex pairs for which an algorithm for the Maximum s-t flow is then executed. More recently, [FNY+20] gave an algorithm with $\tilde{O}(m + nk^3)$ running time for undirected graphs and $\tilde{O}(mk^2)$ time for directed graphs, and [SY22] provided an algorithm with running time $\tilde{O}\left(m^{1+o(1)} \cdot 2^{O(k^2)}\right)$ for undirected graphs. We note that in the worst-case setting where $k = \Omega(n)$, the running times of all these algorithms are at least as high as $\Theta(mn)$.

1.2 Our Results and Techniques

We note that, if a graph G is a complete graph, then it does not have any vertex-cuts. Given a graph G, we can check whether this is the case in time $O(|E(G)|)$. Therefore, in the statements of our results we assume that this is not the case. Our main result is summarized in the following theorem.

Theorem 1.1 *There is a randomized algorithm, that, given a simple undirected n-vertex and m-edge graph G with integral weights $0 \leq w(v) \leq W$ on vertices $v \in V(G)$, such that G is not a complete graph, returns a vertex-cut (A, B, C) in G, such that, with probability at least $1 - 1/n$, cut (A, B, C) is a global minimum vertex-cut in G. The running time of the algorithm is:*

$$O\left(\min\{m^{3/2+o(1)}, mn^{0.99+o(1)}\} \cdot (\log W)^{O(1)}\right).$$

We now provide a high-level overview of the techniques used in the proof of Theorem 1.1, where we mostly focus on the algorithm with running time $O\left(mn^{0.99+o(1)} \cdot (\log W)^{O(1)}\right)$, the most technically challenging part of our result. The high-level intuitive description provided here is somewhat oversimplified and omits some technical details, since we prioritize clarity of exposition over precision in this high-level overview.

Let G be the input simple undirected n-vertex and m-edge graph with integral weights $0 \leq w(v) \leq W$ on its vertices. In order to simplify the discussion in this section, we assume that $W \leq \text{poly}(n)$. For simplicity, we assume that all vertex weights are non-zero; we later show a simple transformation that allows us to achieve this property without changing the problem. Throughout, we use a parameter $0 < \epsilon < 1$. We fix a global minimum vertex-cut (L, S, R), and we assume w.l.o.g. that $w(L) \leq w(R)$. We denote by $\mathsf{OPT} = w(S)$ the value of the optimal solution. It appears that the most difficult special case of the problem is where $|L| \leq n^{\epsilon}$ and the graph G is dense (e.g., $m \geq n^{1.98}$). We start by providing simple algorithms for other special cases, that essentially reduce the problem to this hard special case.

Algorithm Alg_1: when $|S|$ is small. Our first algorithm, Alg_1, is designed to work in the regime where $|S| \leq |L| \cdot n^{1-\epsilon}$. The algorithm for this special case follows the paradigm that was introduced by [LNP+21] in the context of computing global minimum vertex-cut in unweighted graphs. We start by carefully subsampling a subset T of vertices of G in a way that ensures that, with probability at least $\Omega(1/n^{1-\epsilon})$, $|T \cap L| = 1$ and $T \cap S = \emptyset$ holds. If the sampled set T of vertices has these properties, then we say that the sampling algorithm is *successful*. We then use the Isolating Cuts Lemma of [LP20, AKT21b], combined with the recent almost linear-time algorithm for directed Maximum s-t Flow of [CKL+22, vdBCP+23] (see also Theorem 2.7) to compute, in time $O(m^{1+o(1)})$, for every vertex $v \in T$, a minimum vertex-cut separating v from $T \setminus \{v\}$ in G. Lastly, we return the smallest of the resulting cuts. It is immediate to verify that, if the sampling algorithm is successful, then the cut returned by the algorithm is indeed a global minimum vertex-cut. By repeating the algorithm $\tilde{O}(n^{1-\epsilon})$ times, we ensure that with high probability, in at least one of its executions, the sampling algorithm is successful. The total running time of the algorithm for this special case is $O(mn^{1-\epsilon+o(1)})$.

Algorithm Alg_2: when $|S|$ is large and G is not dense. Our second algorithm, Alg_2, is designed for the regime where $|S| \geq n^{1-\epsilon} \cdot |L|$ and $m \leq n^{2-2\epsilon}$. In this case, we let T be the set of all vertices of G whose degree is at least $n^{1-\epsilon}$. Since $m \leq n^{2-2\epsilon}$, it is easy to verify that $|T| \leq 2n^{1-\epsilon}$. Our main observation is that at least one vertex of T must lie in L. Indeed, it is easy to verify that every vertex of S must have an edge connecting it to some vertex of L, and so the total number of edges incident to the vertices of L is at least $|S|$. Since $|S| \geq n^{1-\epsilon} \cdot |L|$, at least one vertex in L must have degree at least $n^{1-\epsilon}$, so it must lie in T. Next, we provide a random procedure that receives as input a vertex $x \in V(G)$, and returns another vertex $y_x \in V(G)$, such that, if $x \in L$, then with a sufficiently high probability $y_x \in R$ holds. The procedure uses the fact that, if $x \in L$, then the total weight of all vertices of S that are not neighbors of x is bounded by $w(L) \leq w(R)$. Vertex y_x is then selected uniformly at random from $V(G) \setminus (\{x\} \cup N_G(x))$, and, if $x \in L$, then it is guaranteed to lie in R with a constant probability. Lastly, for every vertex $x \in T$, we compute a minimum vertex-cut separating x from y_x in G using the almost-linear algorithm of [CKL$^+$22, vdBCP$^+$23] for directed Maximum s-t Flow (see also Corollary 2.6), and return the cut of the smallest weight from among all the resulting cuts. Since at least one vertex $x \in T$ must lie in L, with a sufficiently high probability $y_x \in R$ must hold, and in this case, the cut that the algorithm returns is indeed a minimum vertex-cut.

By combining Algorithms Alg_1 and Alg_2, we obtain an algorithm for global Minimum Vertex-Cut in non-dense graphs: specifically, if $m \leq n^{2-2\epsilon}$, then the running time of the resulting algorithm is $O\left(mn^{1-\epsilon+o(1)}\right)$. We note that this algorithm can be used directly to obtain a randomized algorithm for global minimum vertex-cut with running time $O\left(m^{3/2+o(1)}\right)$; see Section 3.5 for details. We use this algorithm with $\epsilon = 1/100$ to obtain an algorithm with running time $O\left(mn^{0.99+o(1)}\right)$ for graphs in which $m \leq n^{1.98}$ holds. From now on it is sufficient to design an algorithm for dense graphs, where $m \geq n^{1.98}$. In the remainder of this exposition, we focus on such graphs, and we set $\epsilon = 1/45$ for this part of the algorithm. Note that Algorithm Alg_1 works well even in dense graphs, provided that $|S| \leq |L| \cdot n^{1-\epsilon}$ holds. From now on we assume that this is not the case, so in particular, $|L| \leq \frac{|S|}{n^{1-\epsilon}} \leq n^{\epsilon}$. We denote our algorithm for this remaining special case, that we describe below, by Alg_3.

A good set of terminals. At a very high level, we employ the notion of *a good set of terminals*, that was (implicitly) introduced by [LNP$^+$21], in their algorithm for global Minimum Vertex-Cut in unweighted graphs. Recall that we have fixed a global minimum vertex-cut (L, S, R), where $|L| \leq n^{\epsilon}$. For a vertex set $T \subseteq V(G)$, we say that it is a *good set of terminals* if $|T \cap L| = 1$ and $T \cap R \neq \emptyset$ hold, and, additionally, if we denote by x the unique vertex in $T \cap L$, then $T \cap S \subseteq N_G(x)$. For every vertex $x \in T$, we denote by $T_x = T \setminus (\{x\} \cup N_G(x))$. We say that (x, T) is a *good pair*, if T is a good set of terminals, and x is the unique vertex in $L \cap T$. Notice that, if (x, T) is a good pair, then $T_x \subseteq R$ and $T_x \neq \emptyset$, and so any minimum vertex-cut separating x from T_x in G is a global minimum cut.

The algorithm of [LNP$^+$21] starts by selecting a set T of vertices at random, where every vertex is added to T with probability roughly $1/|L|$. It is not hard to show that, if the graph G is unweighted, then with a reasonably high probability, the resulting set T is a good set of terminals, and moreover, $|T| \leq \tilde{O}(n/|L|)$. For every vertex $x \in T$, let G_x be the graph obtained from G by unifying the vertices of T_x into a destination vertex t_x. For every vertex $x \in T$, the algorithm of [LNP$^+$21] computes the value of the minimum x–t_x vertex cut in G_x, and then returns the smallest of the resulting cuts. In order to do so efficiently, they first *sparsify* each such graph G_x, by cleverly employing sparse recovery sketches. With a sufficiently high probability[2], each resulting sparsified graph $\tilde{G}_x$ contains at most $\tilde{O}(\mathsf{OPT} \cdot |L|)$ edges. Applying the almost-linear time algorithm for maximum s-t flow of [CKL$^+$22, vdBCP$^+$23] to

[2]In fact they provide a different simple algorithm for the special case where S contains few low-degree vertices, and the above bound only holds for the remaining case; we ignore these technical details in this informal exposition.

each such graph $\tilde{G}_x$ for $x \in T$ then yields an algorithm for computing global minimum cut in G, whose running time, with high probability, is bounded by $O\left((\mathsf{OPT} \cdot |L|)^{1+o(1)} \cdot |T|\right) \leq O\left(\mathsf{OPT} \cdot n^{1+o(1)}\right)$. Finally, it is easy to see that, in the unweighted setting, the degree of every vertex in G must be at least OPT, so $m \geq \Omega(\mathsf{OPT} \cdot n)$ must hold, leading to the bound of $O\left(m^{1+o(1)}\right)$ on the running time of their algorithm.

Our algorithm also follows this high-level scheme: we use a randomized procedure, that is adapted to vertex-weighted graphs, in order to select a subset $T \subseteq V(G)$ of vertices, so that, with a reasonably high probability, T is a good set of terminals. Then for every terminal $x \in T$, we compute the value of the minimum x-t_x cut in the corresponding graph G_x, and output the smallest such value. But unfortunately, the algorithm of [LNP+21] can no longer be used in order to sparsify each such graph G_x in the weighted setting, and it is not clear how to extend it to this setting. Among other problems, in the weighted setting, the value OPT of the optimal solution may be much higher than n, and we can no longer claim that the degree of every vertex in G is at least OPT. Additionally, in order to ensure that the set T of vertices is a good set of terminals with a sufficiently high probability, we need to sample vertices into T with probability that is proportional to their weight, and other parts of the analysis of the algorithm of [LNP+21] do not work in this setting. We now proceed to describe our algorithm in more detail.

We start by computing the *split graph* G' corresponding to graph G, that is a standard step in transforming cut and flow problems in undirected vertex-capacitated graphs to directed edge-capacitated graphs. The set of vertices of G' is $V(G') = \left\{v^{\text{in}}, v^{\text{out}} \mid v \in V(G)\right\}$; we refer to v^{in} and v^{out} as the *in-copy* and the *out-copy* of v. Given a vertex set $X \subseteq V(G)$, we will also denote by $X^{\text{in}} = \left\{x^{\text{in}} \mid x \in X\right\}$, $X^{\text{out}} = \left\{x^{\text{out}} \mid x \in X\right\}$, and $X^* = X^{\text{in}} \cup X^{\text{out}}$. We denote by $V^{\text{in}} = \left\{v^{\text{in}} \mid v \in V(G)\right\}$ and by $V^{\text{out}} = \left\{v^{\text{out}} \mid v \in V(G)\right\}$. The set $E(G')$ of edges of G' is partitioned into two subsets: the set of *special edges*, that contains, for every vertex $v \in V(G)$, the special edge $e_v = (v^{\text{in}}, v^{\text{out}})$ of capacity $c(e_v) = w(v)$, and the set of *regular edges*, that contains, for every edge $(u, v) \in E(G)$, the regular edges $(u^{\text{out}}, v^{\text{in}})$ and $(v^{\text{out}}, u^{\text{in}})$, whose capacities are set to a value $W_{\max} > 4n \cdot W$, that is an integral power of 2.

For every vertex $x \in T$, let G'_x be the graph that is obtained from G' by adding a destination vertex t, and, for every vertex $y \in T_x$, adding a regular edge (y^{in}, t) of capacity $W_{\max}$ to the graph. It is immediate to verify that the value of the minimum x-t_x vertex-cut in G_x is equal to the value of the minimum x^{out}-t edge-cut in G'_x, which, in turn, from the Max-Flow / Min-Cut theorem, is equal to the value of the maximum x^{out}-t flow in G'_x.

Our algorithm computes, for every vertex $x \in T$, a value c_x, that is at least as high as the value of the maximum x^{out}-t flow in G'_x. Additionally, if (x, T) is a good pair, then our algorithm guarantees that, with a sufficiently high probability, c_x is equal to the value of the maximum x^{out}-t flow in G'_x (which, in turn, must be equal to OPT). Let $x \in T$ be the vertex for which c_x is the smallest, and let y be any vertex in T_x. If T is a good set of terminals, then the value of the global minimum cut in G is then guaranteed, with a sufficiently high probability, to be equal to the value of the minimum vertex-cut separating x from y in G, which, in turn, is equal to c_x. Computing the minimum x-y vertex-cut in G then yields the global minimum vertex-cut with a sufficiently high probability.

At the heart of the algorithm is our main technical subroutine, that receives as input a set $T \subseteq V(G)$ of vertices and a vertex $x \in T$, together with two copies of the graph G'_x in the adjacency-list representation. The algorithm outputs a value c_x, that is at least as high as the value of the maximum x^{out}-t flow in G'_x. Moreover, if (x, T) is a good pair, then with probability at least $1/2$, c_x is guaranteed to be equal to the value of the maximum x^{out}-t flow in G'_x. In the remainder of this exposition, we focus on the algorithm for this subroutine. For convenience, we denote G'_x by G' and $x^{\text{out}} \in V(G')$ by s. We also denote by OPT_x the value of the maximum s-t flow in G'. We note that, since $|T|$ may be

as large as $\Omega(n)$, we need to ensure that the running time of the subroutine is significantly lower than n^2. In particular, if G is sufficiently dense, this running time may need to be lower than $|E(G')|$.

Recall that we have set the parameter $\epsilon = 1/45$. We also use another parameter $n^{7/9} \leq \gamma \leq 2n^{7/9}$, that is an integral power of 2. This setting ensures that $n^{14\epsilon} \leq \gamma \leq 2n^{1-10\epsilon}$. The algorithm for the subroutine consists of $z = O(\operatorname{poly}\log n)$ *phases*. For all $0 \leq i \leq z$, we also use the parameter $M_i = \frac{W'_{\max}}{2^i \cdot \gamma}$, where $W'_{\max}$ is an integral power of 2 that is greater than the largest weight of any vertex in G.

Intuitively, our algorithm maintains an s-t flow f in G', starting with $f = 0$, and then gradually augments it. The *deficit* of the flow f is $\Delta(f) = \mathsf{OPT}_x - \operatorname{val}(f)$. Specifically, for all $1 \leq i \leq z$, we denote by f_i the flow f at the beginning of the ith phase. We ensure that, for all $1 \leq i \leq z$, flow f_i is M_i-integral, and its deficit is bounded by $4nM_i$. Additionally, for technical reasons, we require that, for every vertex $v \in V(G)$ with $w(v) \geq M_i$, if $e = (v^{\text{in}}, v^{\text{out}})$ is the corresponding special edge, then $f_i(e) \leq w(v) - M_i$; in other words, the residual capacity of the edge e remains at least M_i. Assume first that we can indeed ensure these properties. If z is sufficiently large, then the flow f_z has deficit at most $4nM_z < 1$. We can then construct a graph $\hat{G} \subseteq G'$, that only contains edges $e \in E(G')$ with $f(e) > 0$; the number of such edges is bounded by the total running time of our subroutine so far, which, in turn, is significantly less than $\Theta(n^2)$. Since all edge capacities in $\hat{G}$ are integral, the value of the maximum s-t flow in $\hat{G}$ must be $\lceil \operatorname{val}(f_z) \rceil = \mathsf{OPT}_x$. We then compute the value of the maximum s-t flow in $\hat{G}$ and output it as c_x. Before we describe our algorithm for the preprocessing step and for each phase, we need to introduce the notion of *shortcut operations*.

Shortcut Operations. Over the course of the algorithm, we may slightly modify the graph G' by performing *shortcut operations*. In a shortcut operation, we insert an edge (v, t) into G', for some vertex $v \in V(G')$. If the pair (x, T) is not good, then we say that such an operation is always *valid*. Notice that the insertion of the edge may only increase the value of the maximum s-t flow in G'. Otherwise, if the pair (x, T) is good, then we say that the operation is valid only if $v \in S^{\text{out}} \cup R^*$, that is, v is either an out-copy of a vertex of S, or it is an in-copy or an out-copy of a vertex of R. By inspecting the minimum s-t edge-cut in G' (whose corresponding edge set is $\left\{(u^{\text{in}}, u^{\text{out}}) \mid u \in S\right\}$), it is easy to verify that a valid shortcut operation may not change the value of the maximum s-t flow in G' in this case. We ensure that, with a sufficiently high probability, all shortcut operations that the algorithm performs are valid. For clarity, we continue to denote the original graph G' by G', and we use G'' to denote the graph obtained from G' after our algorithm possibly performed a number of shortcut operations. The flow f that our algorithm maintains is in graph G''. Recall that our algorithm is given as input 2 copies of the graph G' in the adjacency-list representation. We use one of these copies to maintain the graph G'', and we use the second copy in order to maintain the residual graph of G'' with respect to the current flow f, that we denote by H.

The Preprocessing Step. The purpose of the preprocessing step is to compute a flow f_1 that can serve as the input to the first phase. Recall that we require that f_1 is M_1-integral, where $M_1 = \frac{W'_{\max}}{2\gamma}$, and that its deficit is at most $4nM_1$. We also require that, for every vertex $v \in V(G)$ with $w(v) \geq M_1$, if $e = (v^{\text{in}}, v^{\text{out}})$ denotes the corresponding edge of G'', then $f_1(e) \leq w(v) - M_1$. Consider the graph $\hat{G} \subseteq G''$, that is obtained from G'' as follows. First, we delete, for every vertex $v \in V(G)$ with $w(v) < M_1$, both copies of v from the graph. Next, for every remaining special edge e, we decrease the capacity of e by at least M_1 and at most $2M_1$ units, so that the new capacity is an integral multiple of M_1. It is easy to verify that, by computing a maximum s-t flow in the resulting flow network $\hat{G}$, we can obtain the flow f_1 with all desired properties. Unfortunately, it is possible that the graph $\hat{G}$ is very dense, so we cannot compute the flow directly. Instead, we *sparsify* $\hat{G}$ first, using the following

two observations, that essentially generalize Equation (2) and Observation 2 of [LNP+21] to arbitrary vertex capacities. Let $N = \left\{v \in V(\hat{G}) \mid (s,v) \in E(\hat{G})\right\}$, and let $N' \subseteq V(G)$ contain all vertices u such that a copy of u lies in N. The first observation is that there is an optimal flow in $\hat{G}$ that does not use any edges that enter the vertices of N, except for the edges that start at s, so all such edges can be deleted. Second, it is not hard to prove that the total number of vertices $v \in S \setminus N'$ of weight at least M_1 is bounded by $2|L|\gamma$. It then follows that, if $u \in V(G)$ has more than $3|L|\gamma$ neighbors in $V(G) \setminus N'$ that have weight at least M_1, then at least one such neighbor must lie in R, and so u may not belong to L. We can then safely add a shortcut edge (v^{out}, t) to both G'' and $\hat{G}$, and we can ignore all other edges leaving v^{out}. These two observations allow us to sparsify the graph $\hat{G}$, obtaining a smaller graph $\hat{G}' \subseteq \hat{G}$, such that, on the one hand, $|E(\hat{G}')| \leq \tilde{O}(n^{2-4\epsilon})$, while on the other hand, all shortcut operations performed so far are valid with a high enough probability, so the value of the maximum s-t flow in $\hat{G}'$ is at least as high as that in $\hat{G}$, and the two flow values are equal if (x,T) is a good pair.

Executing a Single Phase. We now provide the description of the ith phase, for $1 \leq i \leq z$. Recall that we receive as input a flow $f = f_i$ in the current graph G'', that is M_i-integral, and whose deficit is bounded by $4nM_i$. Additionally, for every vertex $v \in V(G)$ with $w(v) \geq M_i$, if $e = (v^{\text{in}}, v^{\text{out}})$ is its corresponding edge in G'', then $f(e) \leq w(v) - M_i$. Therefore, if we denote by $U_i = \{v \in V(G) \mid w(v) \geq M_i\}$, then, for every vertex $v \in U_i$, the residual capacity of the edge $e = (v^{\text{in}}, v^{\text{out}})$ is at least M_i. Throughout, we denote by H the residual flow network of G'' with respect to the current flow f. By the properties of the residual flow network, the value of the maximum s-t flow in H is at least $\mathsf{OPT}_x - \text{val}(f)$ (and it is equal to $\mathsf{OPT}_x - \text{val}(f)$ if (x,T) is a good pair and all shortcut operations so far have been valid). As before, we can consider the graph $\hat{H}$, that is obtained from H by first deleting, for every vertex $u \in V(G) \setminus U_i$, both copies of u, and then reducing, for each remaining forward special edge $(v^{\text{in}}, v^{\text{out}})$, its residual capacity by at least $M_{i+1} = M_i/2$ and at most $2M_{i+1}$, so that it becomes an integral multiple of M_{i+1}. As before, computing a maximum s-t flow in the resulting flow network $\hat{H}$ would yield the flow f_{i+1} with all desired properties. But unfortunately graph $\hat{H}$ may be very dense, and we may not be able to afford to compute the flow directly. As before, we will sparsify this graph first, though the sparsification procedure is much more challenging now. In fact our algorithm may first gradually augment the flow f and add some new shortcut edges, before sparsifying the resulting new graph $\hat{H} \subseteq H$.

The key step in our sparsification procedure is to compute a partition (Q, Q') of the set U_i of vertices of G, so that $|Q' \cap S| < n^{1-4\epsilon}$. Additionally, if $|Q| > n^{1-4\epsilon}$, then we also compute an s-t edge-cut (X^*, Y^*) in the current residual flow network H, that has the following property: the total residual capacity of the edges $E_H(X^*, Y^*)$ is close to the value of the maximum s-t flow in H. In other words, in any maximum s-t flow in H, the edges of $E_H(X^*, Y^*)$ are close to being saturated. Lastly, we ensure that $|E_H(X^*, Y^*)|$ is relatively small, and, for every vertex $v \in Q$, $v^{\text{in}} \in X^*$ and $v^{\text{out}} \in Y^*$ holds. Assume for now that we have computed the partition (Q, Q') of U_i and the cut (X^*, Y^*) as above. Observe that, if some vertex $v \in V(G)$ has more than $2n^{1-4\epsilon}$ neighbors in Q', then, since $|L| \leq n^{\epsilon}$ and $|Q' \cap S| \leq n^{1-4\epsilon}$, at least one of these neighbors must lie in R, and so v may not lie in L. We can then safely add a shortcut edge (v^{out}, t), and ignore all other edges leaving v^{out} in $\hat{H}$. Assume now that $|Q| > n^{1-4\epsilon}$, and let $E' = \{e = (y,x) \in E(H) \mid y \in Y^*, x \in X^*\}$. Consider any maximum s-t flow f^* in H, and denote its value by F. Since the total capacity of all edges in $E_H(X^*, Y^*)$ is close to F, it is easy to see that the total flow that f^* sends on the edges of E' is relatively low. Indeed, if $\mathcal{P}$ is a flow-path decomposition of f^*, and $\mathcal{P}' \subseteq \mathcal{P}$ is the subset of paths containing the edges of E', then every path in $\mathcal{P}'$ must use at least two edges of $E_H(X^*, Y^*)$, so these paths may only carry a relatively small amount of flow. We can then delete the edges of E' from the sparsified graph without decreasing the value of the maximum s-t flow by too much. We use these and other observation in

order to sparsify the graph $\hat{H}$, to obtain a graph $\hat{H}'$ with $|E(\hat{H}')| \leq O(n^{2-4\epsilon+o(1)})$, such that the value of the maximum s-t flow in $\hat{H}'$ is close to that in $\hat{H}$, and all edge capacities in $\hat{H}'$ are integral multiples of M_{i+1}. Computing the maximum s-t flow in $\hat{H}'$ then yields the desired flow f_{i+1}, that serves as the output of the current phase.

Computing the Partition (Q, Q'). The most technically challenging part of our algorithm is computing the partition (Q, Q') of U_i, and, if needed, the cut (X^*, Y^*) in H with the properties described above. Our first step towards this goal is to compute a subgraph $J \subseteq H$ with $s \in J$ and $t \notin J$, such that $|V^{\text{out}} \cap V(J)|$ is small, and additionally, for every vertex $u \notin J$, the total number of edges in set $\{(x, u) \mid x \in V(J)\}$ is relatively small, as is their total capacity in H. Assume first that we have computed such a graph J. We start with an initial partition (Q, Q') of U_i, where Q contains all vertices $v \in U_i$ for which $v^{\text{in}} \in V(J)$ and $v^{\text{out}} \notin V(J)$, and $Q' = U_i \setminus Q$. We show that $|Q' \cap S|$ must be relatively small. Consider now a graph $\tilde{H}$, that is obtained from H, by unifying all vertices of $V(H) \setminus V(J)$ into a new destination vertex t'; we keep parallel edges and discard self-loops. Note that, for every edge $e = (x, y) \in E(H)$ with $x \in V(J)$ and $y \notin V(J)$, there is an edge (x, t') corresponding to e in $\tilde{H}$; we do not distinguish between these two edges. Let F denote the value of the maximum s-t flow in H, and let F' denote the value of the maximum s-t' flow in $\tilde{H}$. Our key observation is that, on the one hand, F' is quite close to F, while, on the other hand, if f' is any maximum s-t' flow in $\tilde{H}$, then the total flow on the edges $\left\{(v^{\text{in}}, v^{\text{out}}) \mid v \in Q \cap S\right\}$ must be close to their capacity. In other words, if we compute any maximum s-t' flow f' in $\tilde{H}$, and a subset $B \subseteq Q$ of vertices, such that the total amount of flow on the edges of $\left\{(v^{\text{in}}, v^{\text{out}}) \mid v \in B\right\}$ is significantly lower than their capacity, then only a small fraction of vertices of B may lie in S; we call such a vertex set $B \subseteq Q$ a *bad batch*. Our algorithm iteratively computes bad batches $B \subseteq Q$ of a sufficiently large cardinality, and then moves all vertices of B from Q to Q'. In order to compute a bad batch, we set up an instance of Min-Cost Maximum s-t' Flow in $\tilde{H}$, by setting the *cost* of every edge (v^{in}, t') corresponding to vertices $v \in Q$ to 1, and setting the costs of all other edges to 0. Once the algorithm terminates, we are guaranteed that Q may no longer contain large bad batches of vertices, or, equivalently, in any maximum s-t' flow in $\tilde{H}$, the vast majority of the edges in $\left\{(v^{\text{in}}, v^{\text{out}}) \mid v \in Q\right\}$ are close to being saturated. If $|Q| \leq n^{1-4\epsilon}$, then we terminate the algorithm with the resulting partition (Q, Q') of U_i. Otherwise, we perform one additional step that computes a cut (X, Y) in the graph $\tilde{H}$, that in turn defines an s-t cut (X^*, Y^*) in H with the desired properties. Note that this step crucially relies on the fact that the graph $\tilde{H}$ is relatively small, which, in turn, follows from the fact that $|V(J) \cap V^{\text{out}}|$ is relatively small, and so is the number of edges $(x, y) \in E(H)$ with $x \in V(J)$ and $y \notin V(J)$.

Computing the Subgraph J. Recall that, in order to compute the partition (Q, Q') as described above, we need first to compute a subgraph $J \subseteq H$ with $s \in J$ and $t \notin J$, such that $|V^{\text{out}} \cap V(J)|$ is small, and additionally, for every vertex $u \notin J$, the total number of edges in set $\{(x, u) \mid x \in V(J)\}$ is relatively small, and so is their total capacity in H. We do not compute this subgraph directly. Instead, we perform a number of iterations. In every iteration, we either add a shortcut edge and augment the current flow f by a significant amount; or we compute the subgraph J of the current residual network H with the desired properties and terminate the algorithm. We note that, as we add shortcut edges, and as the flow f is augmented, the residual flow network H is updated accordingly, and the graph J that our algorithm eventually computes is a subgraph of the current residual flow network H. Our algorithm for this step relies on the technique of *local flow augmentations*, that was first introduced by [CHI+17] in the context of the Maximal 2-Connected Subgraph problem. The technique was later refined and applied to *unweighted* and *approximation* versions of the global minimum vertex-cut problem [NSY19, FNY+20, CQ21].

We first provide a high-level intuitive description of this technique in the context of past work on

unweighted global minimum vertex-cut, and then describe our approach that extends and generalizes this technique. Our description here is somewhat different from that provided in previous work, as we rephrase it to match the terminology used in this overview. Consider an instance G of the unweighted Global Minimum Vertex-Cut problem, and assume that there is an optimal solution (L, S, R) where $|L|$ is small. Suppose we are given a vertex[3] $v \in L$. We construct a split graph G' as before, and add a destination vertex t to it. Intuitively, our goal is to compute a flow of value $|S|$ from $s = v^{\text{out}}$ to t in G', while we are allowed to add valid shortcut edges to G'. We start with a flow $f = 0$, and then iterate. In every iteration, we perform a DFS search in the residual flow network H of G' corresponding to the current flow f, and, once the search discovers roughly $|L| \cdot |S|$ vertices, we terminate it. Let X be the set of vertices that the search has discovered. We note that, if G is a graph with unit vertex weights, and f is an integral flow, then every vertex of V^{in} has exactly one edge leaving it in H, and the other endpoint of the edge lies in V^{out}. Additionally, every vertex of V^{out} has exactly one incoming edge in H. Therefore, at least half the vertices of X must lie in V^{out}. We select a vertex $u \in X \cap V^{\text{out}}$ uniformly at random, and add a shortcut edge (u, t) to G'. Since $|X| \geq 2|L| \cdot |S|$, with probability at least $\left(1 - \frac{1}{2|S|}\right)$, $u \in S^{\text{out}} \cup R^{\text{out}}$, and so the shortcut operation is valid. Let P be the path connecting s to u that the search computed. We augment f by sending 1 flow unit via the path $P \circ (u, t)$ and terminate the iteration. Lastly, if the DFS search only discovers fewer than $2|L| \cdot |S|$ vertices, then the total number of vertices reachable from s in H is small, and we can compute a maximum s-t flow in H directly. Since the number of iterations is bounded by $\mathsf{OPT} = |S|$, with a reasonably high probability, all shortcut operation are valid, and so we compute the value of the Global Minimum Vertex-Cut correctly.

There are several issues with adapting this algorithm to the vertex-weighted setting. The first issue is that, if G is a graph with arbitrary vertex weights, then the residual flow network H may no longer have the property that every vertex of V^{out} has exactly one incoming edge; the number of such edges may be large. As the result, it is possible that the vast majority of the vertices in the set X discovered by the DFS lie in V^{in}. But we cannot select vertices of V^{in} for shortcut operations, since we are not allowed to add edges connecting vertices of S^{in} to t, and it is possible that $|S| = \Omega(n)$. Therefore, our DFS search needs to ensure that the number of vertices of V^{out} that it discovers is sufficiently large. To achieve this high number of vertices of V^{out}, it may be forced to explore a significantly larger number of vertices of V^{in}, leading to a much higher running time of each iteration. In order to obtain a fast overall running time, we then need to ensure that the number of iterations is sufficiently low, and this, in turn, can be ensured by sending a large amount of flow in each iteration. Therefore, when performing a DFS search in H, we only consider edges whose capacities are quite high, at least $\gamma \cdot M_i$. Once the DFS search terminates with a set X containing a relatively low number of vertices of V^{out}, we could let J be the subgraph of H induced by the vertices of X. This approach could indeed be used to guarantee that $|E(J)|$ is low, but unfortunately it does not ensure the other crucial property that we need: that every vertex $u \in V(H) \setminus V(J)$ has relatively few edges (u', u) with $u' \in V(J)$, and that all such edges have a relatively low capacity.

In order to overcome these difficulties, we expand the scope of the local-search technique. In every iteration of the local-search algorithm, we explore the graph H, starting from the vertex s, as follows. Initially, we let $X = \{s\}$. Whenever we encounter an edge (u, v) with $u \in X$ and $v \notin X$, whose capacity in H is at least $\gamma \cdot M_i$, we add v to X. Additionally, whenever we discover a vertex $a \notin X$, such that, for some integer i, there are at least 2^i edges in set $\{(b, a) \in E(H) \mid b \in X\}$, whose capacity is at least $M_i \cdot \gamma / 2^i$, we add a into X. We show that, for every vertex v that is ever added to X via one of the above rules, there is an s-v flow in $H[X]$ of value $\gamma \cdot M_i$. An iteration terminates once $|X \cap V^{\text{out}}|$

[3]In fact a vertex v is selected from G via some random process, and we are only guaranteed that with some reasonably high probability, $v \in L$; this is similar to our setting where we first select a random set T of vertices that is a good set of terminals with a sufficiently high probability, and then select a random vertex $v \in T$.

is sufficiently large, or once neither of the above rules can be applied to expand X. In the latter case, we let $J = H[X]$, and we show that J has all required properties. In the former case, we select a vertex $v \in X \cap V^{\text{out}}$ uniformly at random, and connect it to t via a shortcut operation. We then compute an s-v flow f' of value $\gamma \cdot M_i$ in graph $H[X]$, which is guaranteed to contain relatively few edges, and augment the current flow f by sending $\gamma \cdot M_i$ flow units from s to t via the flow f', that is extended via the edge (v, t) to reach t. We note that, unlike previous work that used the local-flow technique, where the flow augmentation in every iteration was performed via a single path, we augment the flow f via the flow f' that may be more general.

To summarize, the algorithm for a single phase consists of three steps. In the first step, we compute the subgraph $J \subseteq H$, after possibly augmenting the initial flow and adding new shortcut edges. This step exploits the expanded local-search technique. In the second step we compute the partition (Q, Q') of U_i, and, if required, an s-t cut (X^*, Y^*) in H. The former is done by repeated applications of the algorithm for min-cost maximum flow, and the latter is achieved by computing a minimum s-t cut in an appropriately constructed graph. Lastly, in the third step, we sparsify the residual flow network H, and compute the desired flow f_{i+1}. This step relies on the properties of the partition (Q, Q') and of the cut (X^*, Y^*).

A Subgraph Oracle. The algorithm described above needs an access to a subroutine, that we refer to as the *subgraph oracle*. The oracle is given access to an undirected graph G in the adjacency-list representation, a subset Z of its vertices, and a parameter δ. The oracle must return a partition of $V(G)$ into two subsets Y^ℓ and Y^h such that, for every vertex $v \in Y^h$, the number of neighbors of v that lie in Z is at least roughly $n^{1-\delta}$, while for every vertex $u \in Y^\ell$, the number of such neighbors is at most roughly $n^{1-\delta}$. Additionally, the algorithm must return the set $E' = E_G(Y^\ell, Z)$ of edges, and its running time should be at most $n^{2-\Theta(\delta)}$. Our main technical subroutine, when processing a vertex $x \in T$, performs at most $z \leq \text{poly}\log n$ calls to the oracle, with at most one such call performed per phase. While it is not hard to design an algorithm that computes the partition (Y^ℓ, Y^h) of $V(G)$ with the desired properties by using the standard sampling technique, computing the set E' of edges within the desired running time appears much more challenging. Instead, we design an algorithm that can simultaneously process up to n such subgraph queries in bulk, in time $O(n^{3-\Theta(\delta)})$. This algorithm casts the problem of computing the sets E' of edges for all these queries simultaneously as an instance of the Triangle Listing problem in an appropriately constructed graph, and then uses the algorithm of [BPWZ14] for this problem. Our final algorithm for the Global Minimum Vertex-Cut problem proceeds in z iterations. In each iteration i, for every vertex $x \in T$, we run the algorithm for processing x described above, while recording all random choices that the algorithm makes, until it performs the ith call to the subgraph oracle, and then we terminate it. Once we obtain the ith query to the subgraph oracle from each of the algorithms that process vertices of T, we ask the subgraph oracle all these queries in bulk. Then iteration $(i+1)$ is performed similarly, except that we follow all random choices made in iteration i. Since we now have the response to the ith query to the subgraph oracle, we can continue executing the algorithm for processing each vertex $x \in T$ until it performs the $(i+1)$th such query.

Organization. We start with preliminaries in Section 2. We provide a high-level outline of our algorithm in Section 3, including the complete descriptions of Algorithms Alg_1 and Alg_2, and a description of Algorithm Alg_3, with the proof of our main technical result that implements the main subroutine deferred to subsequent sections. In Section 4 we provide the algorithm for the main subroutine, with the algorithms describing Steps 1, 2 and 3 of a single phase provided in sections 5, 6, 7, respectively.

2 Preliminaries

All logarithms in this paper are to the base of 2, unless stated otherwise. We assume that all graphs are given in the adjacency list representation, unless stated otherwise.

2.1 Graph-Theoretic Notation, Cuts and Flows

Given an undirected graph G and a vertex $v \in V(G)$, we denote by $\delta_G(v)$ the set of all edges incident to v in G, by $\deg_G(v) = |\delta_G(v)|$ the degree of v in G, and by $N_G(v)$ the set of all neighbors of v in G: namely, all vertices $u \in V(G)$ with $(u,v) \in E(G)$. If the graph G is directed, then we denote by $N^-_G(v) = \{u \in V(G) \mid (u,v) \in E(G)\}$, and we refer to $N^-_G(v)$ as the *set of all in-neighbors of* v. Similarly, we denote by $N^+_G(v) = \{u \in V(G) \mid (v,u) \in E(G)\}$ the set of all *out-neighbors* of v. We also denote by $\delta^-_G(v)$ and by $\delta^+_G(v)$ the sets of all incoming and all outgoing edges of v in G, respectively. Finally, we denote the degree of v in G by $\deg_G(v) = |\delta^-_G(v)| + |\delta^+_G(v)|$, and we denote by $N_G(v) = N^+_G(v) \cup N^-_G(v)$; we refer to $N_G(v)$ as the set of all neighbors of v in G. Given a pair Z, Z' of subsets of vertices of G (that are not necessarily disjoint), we denote by $E_G(Z, Z')$ the set of all edges (x,y) with $x \in Z$ and $y \in Z'$; this definition applies to both directed and undirected graphs. We may omit the subscript G when the graph is clear from context.

Vertex-Cuts in Undirected Graphs. Let $G = (V, E)$ be an undirected graph with weights $w(v) \geq 0$ on its vertices $v \in V$. For any subset $X \subseteq V$ of its vertices, we denote the *weight of* X by $w(X) = \sum_{v \in X} w(v)$. A *global vertex-cut*, or just a *vertex-cut*, in G is a partition (L, S, R) of its vertices, such that (i) $L, R \neq \emptyset$; and (ii) no edge of G connects a vertex of L to a vertex of R. The *value*, or the *weight* of the cut is $w(S)$. Whenever we consider global vertex-cuts (L, S, R), we will always assume w.l.o.g. that $w(L) \leq w(R)$. In the Global Minimum Vertex-Cut problem, given an undirected graph G with vertex weights as above, the goal is to compute a global vertex-cut of minimum value; we call such a cut a *global minimum vertex-cut*, and we denote its value by OPT. We note that some graphs may not have any vertex cuts (for example a clique graph). In such a case, we may say that the value of the global minimum vertex-cut is infinite.

Consider now an undirected vertex-weighted graph G as above and a pair s, t of its vertices. We say that a vertex-cut (L, S, R) *separates* s from t, or that (L, S, R) is an *s-t vertex-cut*, if $s \in L$ and $t \in R$ holds. We say that it is a *minimum vertex-cut separating s from t*, if the value of the cut is minimized among all such cuts. Similarly, given a vertex $s \in V(G)$ and a subset $T \subseteq V(G) \setminus \{s\}$ of vertices, we say that a vertex-cut (L, S, R) separates s from T if $s \in L$ and $T \subseteq R$ holds. As before, we say that (L, S, R) is the minimum s-T vertex-cut, or a minimum cut separating s from T if it is a smallest-value cut separating s from T. We define cuts separating two subsets X, Y of vertices similarly. Notice that, if there is an edge connecting a vertex of X to a vertex of Y in G, then G does not contain a vertex-cut separating X from Y; in such a case, we say that the value of the minimum vertex-cut separating X from Y is infinite, and that the cut is undefined.

Edge-cuts in directed graphs. Let $G = (V, E)$ be a directed graph with capacities $c(e) \geq 0$ on its edges $e \in E$. Given a subset $E' \subseteq E$ of edges, we denote by $c(E') = \sum_{e \in E'} c(e)$ the *total capacity of the edges in E'*. An *edge-cut* in G is a partition (X, Y) of the vertices of G into two disjoint non-empty subsets, and the *value* of the cut is $c(E_G(X, Y))$. We say that a cut (X, Y) is a *global minimum edge-cut for G*, if it is an edge-cut of G of minimum value. Given a pair s, t of vertices of G, we say that a cut (X, Y) is an *s-t edge-cut*, or that it is an *edge-cut that separates s from t*, if $s \in X$ and $t \in Y$ holds. We define edge-cuts separating a vertex s from a subset $T \subseteq V \setminus \{s\}$ of vertices, and

edge-cuts separating disjoint subsets $S, T \subseteq V(G)$ of vertices similarly.

Flows in edge-capacitated directed graphs. Let $G = (V, E)$ be a directed graph with capacities $c(e) \geq 0$ on its edges $e \in E$, and let $s, t \in V$ be a pair of vertices of G. An *s-t flow* in G is an assignment of flow values $0 \leq f(e) \leq c(e)$ to every edge $e \in E$, such that, for every vertex $v \in V \setminus \{s, t\}$, the following *flow conservation constraint* holds: $\sum_{e \in \delta^+(v)} f(e) = \sum_{e \in \delta^-(v)} f(e)$. We also require that, if $e \in \delta^-(s) \cup \delta^+(t)$, then $f(e) = 0$. We say that the flow f is *acyclic*, if there is no cycle C in the graph, such that every edge $e \in E(C)$ has $f(e) > 0$. For a value $M > 0$, we say that the flow f is *M-integral*, if, for every edge $e \in E$, $f(e)$ is an integral multiple of M (note that it is possible that $M < 1$ under this definition). The *value* of the flow is $\mathrm{val}(f) = \sum_{e \in \delta^+(s)} f(e)$. A *flow-path decomposition* of an s-t flow f is a collection $\mathcal{P}$ of s-t paths in G, together with a flow value $f(P) > 0$ for every path $P \in \mathcal{P}$, such that $\sum_{P \in \mathcal{P}} f(P) = \mathrm{val}(f)$, and, for every edge $e \in E(G)$, $\sum_{\substack{P \in \mathcal{P}: \\ e \in E(P)}} f(P) \leq f(e)$.

A for a vertex $s \in V$ and a subset $T \subseteq V \setminus \{s\}$ of vertices, we can define the s-T flow f similarly: the only difference is that now the flow conservation constraints must hold for all vertices $v \in V \setminus (T \cup \{s\})$, and the flow-paths decomosition of the flow only contains paths connecting s to vertices of T.

2.2 A Modified Adjacency-List Representation of Graphs

Suppose we are given a directed n-vertex and m-edge graph $G = (V, E)$ with integral capacities $0 < c(e) \leq C_{\max}$ on its edges $e \in E$. In order to make our algorithms efficient, we will sometimes use a slight modification of the standard adjacency-list representation of such graphs, that we refer to as the *modified adjacency-list representation* of G.

In the modified adjacencly-list representation of G, for every vertex $v \in V(G)$, for all $0 \leq i \leq \lceil \log(C_{\max}) \rceil + 1$, there are two linked lists $\mathsf{OUT}_i(v)$ and $\mathsf{IN}_i(v)$, where $\mathsf{OUT}_i(v)$ contains all edges $e \in \delta^+(v)$ with $2^i \leq c(e) < 2^{i+1}$, and $\mathsf{IN}_i(v)$ similarly contains all edges $e \in \delta^-(v)$ with $2^i \leq c(e) < 2^{i+1}$. Notice that, given a directed graph G in the adjacency-list representation, we can compute the modified adjacency-list representation of G in time $O(m \log C_{\max})$. Consider now some index $0 \leq i \leq \lceil \log(C_{\max}) \rceil + 1$, and let $G^{\geq i}$ be the graph obtained from G by deleting edges e with $c(e) < 2^i$. Notice that, if we are given the modified adjacency-list representation of G, then we can use it to simulate the modified adjacency-list representation of $G^{\geq i}$, by simply ignoring the lists $\mathsf{OUT}_{i'}(v)$ and $\mathsf{IN}_{i'}(v)$ for all $v \in V$ and $i' < i$.

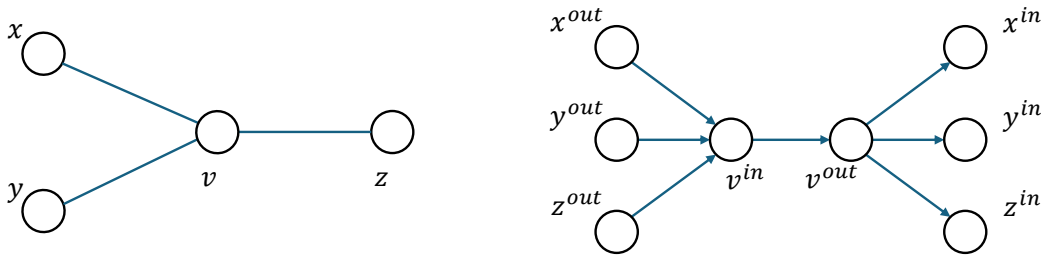
2.3 A Split Graph and Parameters $W'_{\max}$, $W_{\max}$

Given an undirected graph $G = (V, E)$ with integral weights $w(v) \geq 0$ on its vertices $v \in V(G)$, we denote by $W'_{\max}(G)$ the smallest integral power of 2, such that $W'_{\max}(G) > \max_{v \in V} \{w(v)\}$ holds. We let $W_{\max}(G)$ be the smallest integral power of 2 with $W_{\max} \geq 2n \cdot W'_{\max}$. When the graph G is clear from context, we denote $W'_{\max}(G)$ and $W_{\max}(G)$ by $W'_{\max}$ and $W_{\max}$, respectively. Notice that:

$$W_{\max}(G) \leq 4n \cdot W'_{\max}(G) \leq 8n \cdot \max_{v \in V} \{w(v)\}. \tag{1}$$

A *split graph* G' corresponding to G is is defined as follows. The set of vertices of G' contains two copies of every vertex $v \in V$, an *in-copy* v^{in}, and an *out-copy* v^{out}, so $V(G') = \left\{v^{\mathrm{in}}, v^{\mathrm{out}} \mid v \in V\right\}$. The set of edges of G' is partitioned into two subsets: the set E^{spec} of *speical* edges, and the set E^{reg} of *regular edges*. For every vertex $v \in V(G)$, the set E^{spec} of special edges contains the edge $e_v = (v^{\mathrm{in}}, v^{\mathrm{out}})$, whose capacity $c(e_v)$ is set to be $w(v)$; we refer to e_v as the *special edge representing*

v. Additionally, for every edge $e = (x, y) \in E(G)$, we add two regular edges representing e: the edge $(x^{\text{out}}, y^{\text{in}})$, and the edge $(y^{\text{out}}, x^{\text{in}})$. The capacities of both these edges are set to $W_{\max}(G)$. See Figure 2 for the transformation of the neighborhood of a vertex $v \in V(G)$. Given any subset $Z \subseteq V(G)$ of vertices of G, we denote by $Z^{\text{in}} = \{v^{\text{in}} \mid v \in Z\}$, $Z^{\text{out}} = \{v^{\text{out}} \mid v \in Z\}$, and $Z^* = Z^{\text{in}} \cup Z^{\text{out}}$. We also denote by $V^{\text{in}} = \{v^{\text{in}} \mid v \in V(G)\}$ and $V^{\text{out}} = \{v^{\text{out}} \mid v \in V(G)\}$.



(a) Before (b) After

Figure 2: The local transformation for vertex v for computing the split graph.

Observe that, for every pair $s, t \in V(G)$ of vertices, the value of minimum s-t vertex-cut in G is equal to the value of the minimum s^{out}–t^{in} edge-cut in G', which, in turn, is equal to the value of the maximum s^{out}–t^{in} flow in G' from the Max-Flow/Min-Cut theorem. In particular, the value of the global minimum vertex-cut in G is equal to maximum value of the s^{out}–t^{in} flow in G', over all vertex pairs $s, t \in G$. We will use the following simple observation.

Observation 2.1 *Let G be an undirected graph with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V(G)$, and let $s, t \in V(G)$ be a pair of distinct vertices that are not connected by an edge in G. Consider the corresponding split graph G', and let (S, T) is a minimum s^{out}-t^{in} edge-cut in G'. Then there is a deterministic algorithm that, given a cut (S, T) in G' with the above properties, computes, in time $O(|E(G)|)$ a minimum s-t vertex-cut (X, Y, Z) in G.*

Proof: From the definition of the parameter $W_{\max}$, we can assume that every edge in $E_G(S, T)$ is a special edge. If $s^{\text{in}} \in T$, then we move s^{in} from T to S. This move does not increase the capacity of the cut, since the only edge leaving s^{in} in G' is the special edge $(s^{\text{in}}, s^{\text{out}})$. Next, for every vertex $v \in V(G)$ with $v^{\text{in}} \in T$ and $v^{\text{out}} \in S$, we move v^{out} from S to T. Note that this move does not increase the capacity of the cut, since the only edge entering v^{out} in G' is the special edge $(v^{\text{in}}, v^{\text{out}})$. Once all such vertices are processed, we are guaranteed that, for every vertex $v \in V(G)$, if $v^{\text{in}} \in T$, then $v^{\text{out}} \in T$ holds. We let X contain all vertices $v \in V(G)$ with $v^{\text{in}}, v^{\text{out}} \in S$ and Z contain all vertices $v \in V(G)$ with $v^{\text{in}}, v^{\text{out}} \in T$. All remaining vertices are added to Y. Notice that, for every vertex $v \in Y$, $v^{\text{in}} \in S$ and $v^{\text{out}} \in T$ holds. Therefore, $w(Y) = \sum_{e \in E_{G'}(S,T)} c(e)$. As observed already, the value of the minimum s^{out}-t^{in} edge-cut in G' is equal to the value of the minimum s-t vertex-cut in G, so the observation follows. □

Lastly, observe that $|V(G')| = 2|V(G)|$ and $|E(G')| \leq O(|E(G)|)$; moreover, there is an algorithm that, given G, computes G' in time $O(|E(G')|)$.

2.4 Properties of the Global Minimum Vertex-Cut

In this subsection we establish some useful properties of a global minimum vertex-cut in undirected graphs. Consider a vertex-weighted graph G, let (L, S, R) be a global minimum vertex-cut in G, and

let $x \in L$ be any vertex. The following simple claim shows that, in a sense, almost all vertices of S must be neighbors of x in G. In other words, the total weight of all vertices of S that are not neighbors of x is bounded by $w(L)$.

The proof of the claim needs the definition of flows in undirected vertex-capacitated graphs. For convenience, we define such flows via their flow-path decompositions. Specifically, suppose we are given an undirected graph G with weights $w(v) \geq 0$ on its vertices $v \in V$, and two special vertices s and t. An *s-t flow* in G consists of a collection $\mathcal{P}$ of paths, each of which connects s to t, and, for every path $P \in \mathcal{P}$, a flow value $f(P)$. We require that, for every vertex $v \in V(G) \setminus \{s,t\}$, $\sum_{\substack{P \in \mathcal{P}: \\ v \in P}} f(P) \leq w(P)$ holds. We refer to the paths in $\mathcal{P}$ as the *flow-paths* of f. From the Max-Flow/Min-cut Theorem, the value of the maximum s-t flow in a vertex-weighted graph G is equal to the value of the minimum s-t vertex-cut.

The following claim can be thought of as a generalization of Proposition 3.2 in [LNP+21] from unit vertex weights to general weights.

Claim 2.2 *Let G be a simple undirected graph with integral weights $w(v) > 0$ on its vertices $v \in V(G)$, and let (L, S, R) be a global minimum vertex-cut in G. Let $x \in L$ be any vertex, and denote by $S' = S \setminus N_G(x)$. Then $w(S') \leq w(L)$ must hold.*

Proof: Let $t \in R$ be any vertex. From the Max-Flow/Min-Cut theorem, there is a flow f in G from x to t, that respects all vertex capacities (that are equal to their weights), whose value is $\mathsf{OPT} = w(S)$. Let $\mathcal{P}$ be the corresponding collection of flow-paths. We partition $\mathcal{P}$ into two sets: set $\mathcal{P}' \subseteq \mathcal{P}$ contains all paths P, whose second vertex lies in S, and $\mathcal{P}'' = \mathcal{P} \setminus \mathcal{P}'$ contains all remaining flow-paths. Notice that, for every path $P \in \mathcal{P}''$, the second vertex on P must lie in L, and so $\sum_{P \in \mathcal{P}''} f(P) \leq w(L)$ must hold. Therefore, $\sum_{P \in \mathcal{P}'} f(P) \geq \mathsf{OPT} - w(L) = w(S) - w(L)$. For each flow-path $P \in \mathcal{P}'$, the second vertex on P lies in $S \cap N_G(x)$, so $w(S \cap N_G(x)) \geq w(S) - w(L)$, and $w(S') = w(S) - w(S \cap N_G(x)) \leq w(L)$. □

We obtain the following immediate corollary of the claim.

Corollary 2.3 *Let G be a simple undirected graph with integral weights $0 < w(v) \leq W$ on its vertices $v \in V(G)$, and let (L, S, R) be a global minimum vertex-cut in G. Let $x \in L$ be any vertex, let $\gamma > 0$ be a parameter, and denote by S'_γ the set of all vertices $v \in S \setminus N_G(x)$ with $w(v) \geq \frac{W}{\gamma}$. Then $|S'_\gamma| \leq |L| \cdot \gamma$ must hold.*

Proof: Let $S' = S \setminus N_G(x)$. Observe first that $w(L) \leq |L| \cdot W$ must hold, and, from Claim 2.2, $w(S') \leq w(L) \leq |L| \cdot W$ holds. Since $S'_\gamma \subseteq S'$, and since, for every vertex $v \in S'_\gamma$, $w(v) \geq \frac{W}{\gamma}$ holds, we get that:

$$|S'_\gamma| \leq \frac{w(S')}{W/\gamma} \leq |L| \cdot \gamma.$$

□

2.5 Fast Algorithms for Flows and Cuts

Our algorithm relies on the recent breakthrough almost linear time algorithm for Minimum Cost Maximum s-t Flow [CKL+22, vdBCP+23]. The algorithm of [vdBCP+23] first computes a fractional solution to the problem, whose cost is close to the optimal one, and then rounds the resulting flow via the Link-Cut Tree data structure [ST83, KP15] in order to compute the optimal flow. In particular,

the flow that they obtain is integral, provided that all edge capacities are integral as well (see Lemma 4.1 in [vdBCP+23]). This result is summarized in the following theorem (see Theorem 1.1 and Lemma 4.1 in [vdBCP+23]).

Theorem 2.4 ([vdBCP+23] (see also [CKL+22])) *There is a deterministic algorithm, that receives as input a directed m-edge graph G with integral capacities $1 \leq u(e) \leq U$ and integral costs $1 \leq c(e) \leq C$ on edges $e \in E(G)$, together with two vertices $s, t \in V(G)$. The algorithm computes a minimum-cost maximum s-t flow f in G, such that f is integral. The running time of the algorithm is $O\left(m^{1+o(1)} \cdot \log U \cdot \log C\right)$.*

It is well known that, given a maximum s-t flow in a flow network G, one can compute a minimum s-t cut in G in time $O(m)$. In order to do so, we compute a residual flow network H of G with respect to f, and then compute the set S of all vertices of H that are reachable from s in H. Let $T = V(G) \setminus S$. Then, by the Max-Flow / Min-Cut theorem [FF56], (S, T) is a minimum s-t cut in G. By combining this standard algorithm with the algorithm from Theorem 2.4, we obtain the following immediate corollary.

Corollary 2.5 *There is a deterministic algorithm, that receives as input a directed m-edge graph G with integral capacities $1 \leq u(e) \leq U$ on edges $e \in E(G)$, together with two distinct vertices $s, t \in V(G)$. The algorithm computes a minimum s-t edge-cut (S, T) in G. The running time of the algorithm is $O\left(m^{1+o(1)} \cdot \log U\right)$.*

Assume now that we are given an undirected graph $G = (V, E)$ with integral weights $w(v)$ on its vertices $v \in V$, and two disjoint vertex subsets $S, T \subseteq V$. We can use the algorithm from Corollary 2.5 in order to compute a minimum S-T vertex-cut in G, as follows. Observe first that, if there is an edge connecting a vertex of S to a vertex of T in G, or if $S \cap T \neq \emptyset$, then the minimum S-T vertex-cut is undefined; we assume that this is not the case. Let $\hat{G}$ be the graph obtained from G by setting the weights of the vertices in $S \cup T$ to ∞, and then adding vertices s and t of unit weight, where s connects with an edge to every vertex in S, and t connects with an edge to every vertex in T. It is immediate to verify that, if (X, Y, Z) is a minimum s-t vertex-cut in $\hat{G}$, then $(X \setminus \{s\}, Y, Z \setminus \{t\})$ is a minimum S-T vertex-cut in G and vice versa. We then compute a split graph $\hat{G}'$ corresponding to $\hat{G}$, and use the algorithm from Corollary 2.5 to compute the minimum s^{out}-t^{in} edge-cut (S, T) in $\hat{G}'$. Lastly, we use the algorithm from Observation 2.1 to compute a minimum s-t vertex-cut (X, Y, Z) in $\hat{G}$, and then output $(X \setminus \{s\}, Y, Z \setminus \{t\})$ as a minimum S-T vertex-cut in G. We summarize this algorithm in the following corollary.

Corollary 2.6 *There is a deterministic algorithm, that receives as input an undirected m-edge graph G with integral weights $1 \leq w(v) \leq W$ on vertices $v \in V(G)$, together with two disjoint subsets $S, T \subseteq V(G)$ of vertices. The algorithm computes a minimum S-T vertex-cut (X, Y, Z) in G. The running time of the algorithm is $O\left(m^{1+o(1)} \cdot \log W\right)$.*

2.6 Computing Minimum Cuts via Isolating Cuts

The Isolating Cuts lemma was introduced independently in [LP20, AKT21b], and found many applications since (see, e.g. [LP21, CQ21, MN21, LNP+21, AKT21a, LPS21, Zha22, AKT22, CLP22, LNPS23, FPZ23, HHS24, HLRW24, ACD+24]). We use the following theorem, that generalizes the version of the Isolating Cuts lemma from [LNP+21] for unweighted graphs to graphs with arbitrary vertex weights. The proof closely follows the arguments from [LNP+21], and is provided in Section A of Appendix for completeness.

Theorem 2.7 (Isolating Cuts) *There is a deterministic algorithm, that, given as input an undirected m-edge graph G with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V(G)$, together with a subset $T \subseteq V(G)$ of vertices of G, computes, for every vertex $v \in T$, the value of the minimum vertex-cut separating v from $T \setminus \{v\}$. The running time of the algorithm is $O\left(m^{1+o(1)} \cdot \log W\right)$.*

2.7 The Chernoff Bound and a Useful Inequality

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

Lemma 2.8 (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 $t > 2e\mu$,*

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

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

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

We also use an inequality summarized in the following simple claim.

Claim 2.9 *For all $r \geq 2$, $\left(1 - \frac{1}{r}\right)^{r-1} \geq \frac{1}{e^2}$ holds.*

Proof: From Taylor's expansion of function $\ln(1+x)$, it is easy to see that, for all $0 \leq \epsilon \leq 1/2$, $\ln(1-\epsilon) > -\epsilon - \epsilon^2$ holds. Therefore:

$$\ln\left(1 - \frac{1}{r}\right) \geq -\frac{1}{r} - \frac{1}{r^2} \geq -\frac{2}{r}.$$

We then get that:

$$\ln\left(\left(1 - \frac{1}{r}\right)^{r-1}\right) \geq -\frac{2(r-1)}{r} \geq -2,$$

and:

$$\left(1 - \frac{1}{r}\right)^{r-1} \geq \frac{1}{e^2}.$$

□

2.8 The Degree Estimation Problem

We will sometimes use a natural randomized algorithm in order to efficiently estimate the degrees of some vertices in a subgraph of a given graph G. Specifically, suppose we are given a graph G, two subsets Z, Z' of its vertices, and a threshold τ. Our goal is to compute a subset $A \subseteq Z'$ of vertices, such that, on the one hand, every vertex $v \in A$ has at least τ neighbors that lie in Z, while, on the other hand, every vertex $v \in Z'$ with $|N_G(v) \cap Z| \gg \tau$ is in A. We now define the Undirected Degree Estimation problem more formally and provide a simple and natural randomized algorithm for it.

Definition 2.10 (Undirected Degree Estimation problem.) *In the* Undirected Degree Estimation problem*, the input is a simple undirected n-vertex graph $G = (V, E)$ with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V$, given in the adjacency-list representation, two (not necessarily disjoint) subsets $Z, Z' \subseteq V(G)$ of vertices of G, a threshold $1 \leq \tau \leq n$. The output to the problem is a subset $A \subseteq Z'$ of vertices. We say that an algorithm for the problem* errs*, if either (i) for some vertex $v \in A$, $|N_G(v) \cap Z| < \tau$; or (ii) for some vertex $v' \in Z' \setminus A$, $|N_G(v) \cap Z| \geq 1000\tau \log n \cdot \log(W_{\max}(G))$.*

The following claim provides a simple algorithm for the Degree Estimation problem, using standard techniques. The proof is provided in Appendix B.1.

Claim 2.11 (Algorithm for the Undirected Degree Estimation Problem) *There is a randomized algorithm for the Undirected Degree Estimation problem, whose running time is $O\left(\frac{n \cdot |Z| \cdot \log n \cdot \log(W_{\max})}{\tau}\right)$. The probability that the algorithm errs is at most $\frac{1}{n^{10} \cdot \log^6(W_{\max})}$, where $W_{\max} = W_{\max}(G)$.*

We will also use the following directed version of the degree estimation problem.

Definition 2.12 (Directed Heavy Degree Estimation Problem) *In the* Directed Heavy Degree Estimation problem*, the input is a simple directed n-vertex graph $G = (V, E)$ with capacities $c(e) > 0$ on edges $e \in E$, given in the adjacency-list representation, two (not necessarily disjoint) subsets $Z, Z' \subseteq V(G)$ of vertices of G, a threshold $\tau \geq 1$, and two other parameters $c^* > 0$ and W that is greater than a large enough constant. For a vertex $v \in Z$, denote by E_v the set of all edges $(v, u) \in E$ whose capacity is at least c^*, such that $u \in Z'$. The output of the problem is a subset $A \subseteq Z$ of vertices. We say that an algorithm for the problem* errs*, if one of the following hold:*

- *either there is some vertex $u \in A$ with $|E_u| < \tau$; or*
- *there is a vertex $u' \in Z \setminus A$ with $|E_u| \geq 1000\tau \log n \cdot \log W$.*

The following claim provides an algorithm for the Directed Heavy Degree Estimation Problem, that is a simple adaptation of the algorithm from Claim 2.11. We provide a proof sketch of this claim in Appendix B.1.

Claim 2.13 (Algorithm for the Directed Heavy Degree Estimation Problem) *There is a randomized algorithm for the Directed Heavy Degree Estimation problem, whose running time is $O\left(\frac{n \cdot |Z'| \cdot \log n \cdot \log W}{\tau}\right)$. The probability that the algorithm errs is at most $\frac{1}{n^{10} \cdot \log^6 W}$.*

2.9 The Heavy Vertex Problem

We will also sometimes use an algorithm for the Heavy Vertex problem, that is very similar to the Directed Degree Estimation problem. The main difference is that goal of the algorithm is to either (i) compute a vertex $v \in Z'$, such that there are at least τ vertices $u_1, \ldots, u_\tau \in Z$, where for all $1 \leq j \leq \tau$, an edge (u_j, v) of capacity at least c^* lies in the graph; or (ii) correctly establish that no such vertex exists. Like in the Directed Degree Estimation problem, in order to ensure that the algorithm is efficient, we will solve the problem approximately. We now define the Heavy Vertex problem formally.

Definition 2.14 (Heavy Vertex problem.) *In the* Heavy Vertex problem*, the input is a simple directed n-vertex graph $G = (V, E)$ with capacities $c(e) > 0$ on edges $e \in E$, given in the adjacency-list representation, two disjoint subsets $Z, Z' \subseteq V(G)$ of vertices of G, an integer $\tau \geq 1$, and two other parameters $c^* > 0$, and $W > 0$ that is greater than a sufficiently large constant. The output*

of the problem is a bit $b \in \{0,1\}$, and, if $b = 1$, a vertex $v \in Z'$, together with τ distinct vertices $u_1, \ldots, u_\tau \in Z$, such that, for all $1 \leq j \leq \tau$, there is an edge (u_j, v) of capacity at least c^* in G. We say that an algorithm for the problem errs*, if it returns $b = 0$, and yet there exists a vertex $v \in Z'$, and a collection $U \subseteq Z$ of vertices with $|U| \geq 1000\tau \log n \log W$, such that, for every vertex $u \in U$, there is an edge (u, v) of capacity at least c^* in G.*

For convenience, we denote the input to the Heavy Vertex problem by (G, Z, Z', τ, c^*, W). The following simple claim provides an algorithm for the Heavy Vertex problem. The proof of the claim is deferred to Section B.2 of Appendix.

Claim 2.15 (Algorithm for the Heavy Vertex problem) *There is a randomized algorithm for the Heavy Vertex problem, whose running time, on input (G, Z, Z', τ, c^*, W), is $O\left(\frac{n \cdot |Z| \cdot \log n \cdot \log W}{\tau}\right)$. The probability that the algorithm errs is at most $1/\left(n^{10} \cdot \log^6 W\right)$.*

3 The Algorithm

In this section we provide our main result, namely an algorithm for the vertex-weighted global minimum vertex cut, proving Theorem 1.1. We defer some of the technical details to subsequent sections.

We assume that we are given a simple undirected n-vertex and m-edge graph G with integral weights $0 \leq w(v) \leq W$ on its vertices. It will be convenient for us to assume that all vertex weights are strictly positive. In order to achieve this, we can modify the vertex weights as follows: for every vertex $v \in V(G)$, we let $w'(v) = n^2 \cdot w(v) + 1$. Let G' be the graph that is identical to G, except that the weight of every vertex v is set to $w'(v)$, and let $W' = n^2 \cdot W + 1$, so for every vertex $v \in V(G)$, $1 \leq w'(v) \leq W'$ holds. Note that, if (L, S, R) is a global minimum vertex cut in G', then it must also be a global minimum vertex-cut in G. Therefore, it is now enough to compute a global minimum vertex-cut in G'. In order to avoid clutter, we will denote G' by G, and the vertex weights $w'(v)$ by $w(v)$ for $v \in V(G)$. From now on we assume that, for every vertex $v \in V(G)$, $1 \leq w(v) \leq W'$, where $W' = n^2 \cdot W + 1$. We define the values $W'_{\max}(G)$ and $W_{\max}(G)$ as in Section 2.3, and denote them by $W'_{\max}$ and $W_{\max}$, respectively, throughout this section. Note that:

$$W_{\max} \leq O(nW') \leq O(n^3 \cdot W). \tag{2}$$

Let $\rho = \lceil \log(W') \rceil$. For all $0 \leq i \leq \rho$, we let $U_i \subseteq V$ denote the set of all vertices v with $2^i \leq w(v) < 2^{i+1}$, so $(U_0, \ldots, U_\rho)$ is a partition of $V(G)$.

Whenever we are given any vertex cut (L, S, R) of G, we assume without loss of generality that $w(L) \leq w(R)$ holds. For all $0 \leq i \leq \rho$, we then denote $L_i = L \cap U_i$, $S_i = S \cap U_i$, and $R_i = R \cap U_i$, so that (L_i, S_i, R_i) is a partition of U_i.

We use a combination of several algorithms, that are designed to handle different edge densities and different tradeoffs between the values $|L_i|$ and $|S_i|$ for the optimal cut (L, S, R), for $0 \leq i \leq \rho$. Each such algorithm must return a pair s, t of vertices of G, together with a value c that is at least as large as the value of the minimu s-t vertex-cut in G. We will ensure that, with high probability, at least one of the algorithms returns a value $c = \mathsf{OPT}$. Taking the smallest value c returned by any of these algorithms then yields the optimal solution value with high probability, and computing the minimum s-t vertex cut for the corresponding pair s, t of vertices provides an optimal global minimum cut in G. We now describe each of the algorithms in turn.

3.1 Algorithm Alg_1: when S_i is Small for Some i

Our first algorithm, Alg_1, is summarized in the following theorem.

Theorem 3.1 *There is a randomized algorithm, that we denote by Alg_1, whose input is a simple undirected n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$, and a parameter $0 < \epsilon < 1$. The algorithm returns two distinct vertices $x, y \in V(G)$ with $(x, y) \notin E(G)$, and a value c that is at least as large as the value of the minimum x-y vertex-cut in G. Moreover, if there is a global minimum vertex-cut (L, S, R) in G, and an integer $0 \leq i \leq \lceil \log W' \rceil$, such that $L_i \neq \emptyset$ and $|S_i| \leq |L_i| \cdot n^{1-\epsilon}$, then, with probability at least $1 - 1/n^4$, $c = \mathsf{OPT}$. The running time of the algorithm is $O\left(mn^{1-\epsilon+o(1)} \cdot \log^2 W'\right)$.*

In the remainder of this subsection, we prove the above theorem. At a high level, Algorithm Alg_1 uses the paradigm of subsampling a set T of vertices of G and then applying the Isolating Cuts Lemma (Theorem 2.7) to T; this paradigm was introduced by [LNP+21] in the context of computing global minimum vertex-cut in unweighted graphs. The following lemma is central to the proof of the theorem.

Lemma 3.2 *There is a randomized algorithm, whose input is a simple undirected n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$, two integral parameters $0 \leq i \leq \lceil \log W' \rceil$ and $0 < \lambda \leq n$, and another parameter $0 < \epsilon < 1$. The algorithm returns two distinct vertices $x, y \in V(G)$ with $(x, y) \notin E(G)$, and a value c that is at least as large as the value of the minimum x-y vertex-cut in G. Moreover, if there is a global minimum vertex-cut (L, S, R) in G with $\lambda \leq |L_i| + |S_i| \leq 2\lambda$, such that $L_i \neq \emptyset$ and $|S_i| \leq |L_i| \cdot n^{1-\epsilon}$, then, with probability at least $1 - 1/n^4$, $c = \mathsf{OPT}$ holds. The running time of the algorithm is $O\left(mn^{1-\epsilon+o(1)} \cdot \log W'\right)$.*

We prove Lemma 3.2 below, after we complete the proof of Theorem 3.1 using it. Let $z = \lceil \log W' \rceil$ and let $z' = \lceil \log n \rceil$. Our algorithm performs $O(z \cdot z')$ iterations, as follows. For all $0 \leq i \leq z$ and for all $0 \leq j \leq z'$, we apply the algorithm from Lemma 3.2 to graph G, with parameter i and $\lambda = 2^j$, and parameter ϵ remaining unchanged; we denote this application of the algorithm from Lemma 3.2 by $\mathcal{A}_{i,j}$, and we denote by $(c_{i,j}, x_{i,j}, y_{i,j})$ its output. We let i^*, j^* be the indices for which the value c_{i^*,j^*} that Algorithm $\mathcal{A}_{i,j}$ returned is the smallest, breaking ties arbitrarily. We then output $(c_{i^*,j^*}, x_{i^*,j^*}, y_{i^*,j^*})$. This completes the description of Algorithm Alg_1.

Since the running time of the algorithm from Lemma 3.2 is $O(mn^{1-\epsilon+o(1)} \cdot \log W')$, the running time of Alg_1 is $O(mn^{1-\epsilon+o(1)}) \cdot O(z \cdot z') \leq O(mn^{1-\epsilon+o(1)} \cdot \log^2 W')$. Moreover, Lemma 3.2 ensures that the value of the minimum x_{i^*,j^*}-y_{i^*,j^*} vertex-cut in G is at most c_{i^*,j^*}.

Assume now that there is a global minimum vertex cut (L, S, R) in G, and an integer $0 \leq i \leq \lceil \log W' \rceil$, such that $L_i \neq \emptyset$ and $|S_i| \leq |L_i| \cdot n^{1-\epsilon}$. Let $0 \leq j \leq z'$ be the unique integer with $2^j \leq |L_i| + |S_i| < 2^{j+1}$. We say that Algorithm $\mathcal{A}_{i,j}$ is *successful*, if its output $(c_{i,j}, x_{i,j}, y_{i,j})$ has the property that $c_{i,j} = \mathsf{OPT}$. From Lemma 3.2, with probability at least $1 - 1/n^4$, Algorithm $\mathcal{A}_{i,j}$ is successful. In this case we are guaranteed that $c_{i^*,j^*} = \mathsf{OPT}$ as well. In order to complete the proof of Theorem 3.1, it is now enough to prove Lemma 3.2.

Proof of Lemma 3.2. The proof follows the high-level idea of [LNP+21], that was introduced in the context of computing global minimum vertex-cuts in unweighted graphs, of subsampling a subset T of vertices of G, and then applying the Isolating Cuts Lemma (Theorem 2.7) to it.

We start by describing an algorithm, that we denote by $\mathcal{A}'$. Our final algorithm will execute $\mathcal{A}'$ several times. Algorithm $\mathcal{A}'$ starts by constructing a random set T of vertices as follows: every vertex $v \in U_i$ is added to T independently with probability $\frac{1}{2\lambda}$. We then use the algorithm from Theorem 2.7 to compute, for every vertex $v \in T$, the value c_v of the minimum vertex-cut separating v from $T \setminus \{v\}$ in

G. We let $x \in T$ be the vertex for which the value c_x is the smallest, breaking ties arbitrarily, and we let y be an arbitrary vertex in $T \setminus \{x\}$. We then return c_x, x and y as the outcome of the algorithm $\mathcal{A}'$. Note that the running time of Algorithm $\mathcal{A}'$ is $O(m^{1+o(1)})$. It is easy to verify that there is an x-y vertex-cut of value c_x in G (the cut separating x from $T \setminus \{x\}$), and the value of the minimum x-y vertex-cut in G is at most c_x. We say that the algorithm $\mathcal{A}$ is *successful* if $c_x = \mathsf{OPT}$; otherwise we say that it is *unsuccessful*. We will use the following claim to analyze algorithm $\mathcal{A}'$.

Claim 3.3 *Suppose that there is a global minimum vertex-cut (L, S, R) in G with $\lambda \leq |L_i| + |S_i| \leq 2\lambda$, such that $L_i \neq \emptyset$ and $|S_i| \leq |L_i| \cdot n^{1-\epsilon}$ holds. Then the probability that Algorithm $\mathcal{A}'$ is successful is at least $\Omega\left(\frac{1}{n^{1-\epsilon}}\right)$.*

Proof: We say that the good Event $\mathcal{E}$ happens if $|T \cap L_i| = 1$ and $|T \cap S_i| = \emptyset$. Assume first that Event $\mathcal{E}$ happened, and let v be the unique vertex in $T \cap L_i$, so $T \setminus \{v\} \subseteq R$. Since (L, S, R) is a minimum vertex-cut separating v from $T \setminus \{v\}$, we get that $c_v = w(S) = \mathsf{OPT}$ must hold. Moreover, for every vertex $u \in T \setminus \{v\}$, c_u is at least as large as the value of some vertex-cut in G, so $c_u \geq \mathsf{OPT}$ holds as well. Therefore, if we denote the outcome of the algorithm by (c_x, x, y), then $c_x = \mathsf{OPT}$ must hold. Since c_x is the value of the minimum vertex-cut separating x from $T \setminus \{x\}$, we get that there is an x-y cut of value OPT. We conclude that, if Event $\mathcal{E}$ happened, then the algorithm is successful. The following observation will then complete the proof of Claim 3.3.

Observation 3.4 $\mathbf{Pr}\left[\mathcal{E}\right] \geq \Omega\left(\frac{1}{n^{1-\epsilon}}\right)$.

Proof: For every vertex $v \in L_i$, we define the event $\mathcal{E}(v)$ that $v \in T$, and T contains no other vertices of $S_i \cup L_i$. Denote $|S_i \cup L_i|$ by z, and recall that $\lambda \leq z \leq 2\lambda$. Then for all $v \in L_i$:

$$\begin{aligned}\mathbf{Pr}\left[\mathcal{E}(v)\right] &= \frac{1}{2\lambda} \cdot \left(1 - \frac{1}{2\lambda}\right)^{z-1} \\ &\geq \frac{1}{2\lambda} \cdot \left(1 - \frac{1}{2\lambda}\right)^{2\lambda-1} \\ &\geq \frac{1}{a\lambda},\end{aligned}$$

for some constant $a \geq 1$. For the first inequality, we have used the fact that $\lambda \leq z \leq 2\lambda$ holds, and the second inequality follows from Claim 2.9.

Recall that $|L_i| \geq \frac{|S_i|}{n^{1-\epsilon}}$, and so $|L_i| \geq \frac{|S_i \cup L_i|}{2n^{1-\epsilon}} \geq \frac{\lambda}{2n^{1-\epsilon}}$ must hold. Since Events $\mathcal{E}(v)$ for $v \in L_i$ are all disjoint, and since event $\mathcal{E}$ only happens if one of the events $\mathcal{E}_v$ for $v \in L_i$ happens, we get that:

$$\mathbf{Pr}\left[\mathcal{E}\right] = \sum_{v \in L_i} \mathbf{Pr}\left[\mathcal{E}_i\right] \geq \frac{|L_i|}{a\lambda} \geq \Omega\left(\frac{1}{n^{1-\epsilon}}\right).$$

□ □

We perform $\Theta(n^{1-\epsilon} \cdot \log n)$ executions of Algorithm $\mathcal{A}'$, and return the triple (c_x, x, y) with smallest value c_x that any execution of the algorithm produced (if $x = y$ or $(x, y) \in E(G)$ holds, then we instead return an arbitrary pair of distinct vertices of G that are not connected by an edge, and the value $c = W_{\max}(G)$). It is easy to verify that, if there is a global minimum vertex-cut (L, S, R) in G with $\lambda \leq |L_i| + |S_i| \leq 2\lambda$, such that $L_i \neq \emptyset$ and $|S_i| \leq |L_i| \cdot n^{1-\epsilon}$, then the probability that at least one of the executions of $\mathcal{A}'$ is successful is at least $1 - 1/n^4$, and in this case, if (c, x, y) is the output of the algorithm, then we are guaranteed that $c = \mathsf{OPT}$ holds. The running time of the entire algorithm is $O\left(m \cdot n^{1-\epsilon+o(1)} \cdot \log W'\right)$. □

3.2 Algorithm Alg_2: when G is not Dense and S is Large

Our second algorithm, Alg_2, will be used in the case where the input graph G is not very dense; we summarize it in the following theorem.

Theorem 3.5 *There is a randomized algorithm, that we denote by Alg_2, whose input is a simple undirected n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$, and a parameter $0 < \epsilon < 1$, such that $m \leq n^{2-2\epsilon}$ holds. The algorithm returns two distinct vertices $x, y \in V(G)$ with $(x, y) \notin E(G)$, and a value c that is at least as large as the value of the minimum x-y vertex-cut in G. Moreover, if there is a global minimum vertex-cut (L, S, R) in G with $|S| \geq n^{1-\epsilon} \cdot |L|$, then, with probability at least $1 - 1/n^4$, $c = \mathsf{OPT}$ holds. The running time of the algorithm is $O\left(mn^{1-\epsilon+o(1)} \cdot \log W'\right)$.*

The remainder of this subsection is dedicated to the proof of Theorem 3.5.

Let T be the set of all vertices $v \in V(G)$ with $\deg_G(v) \geq n^{1-\epsilon}$. Since $\sum_{v \in V(G)} \deg_G(v) = 2m \leq 2n^{2-2\epsilon}$, we get that $|T| \leq \frac{2n^{2-2\epsilon}}{n^{1-\epsilon}} \leq 2n^{1-\epsilon}$. We use the following simple observation.

Observation 3.6 *Suppose there is a global minimum vertex-cut (L, S, R) in G with $|S| \geq n^{1-\epsilon} \cdot |L|$. Then $L \cap T \neq \emptyset$ must hold.*

Proof: Let (L, S, R) be a global minimum vertex-cut in G with $|S| \geq n^{1-\epsilon} \cdot |L|$. We claim that, for every vertex $v \in S$, there must be an edge connecting v to some vertex of L. Indeed, assume otherwise. Let $v \in S$ be any vertex that has no neighbors in L. Consider the partition (L, S', R') of vertices of G with $S' = S \setminus \{v\}$ and $R' = R \cup \{v\}$. It is easy to verify that (L, S', R') is a valid vertex-cut in G, and moreover that $w(S') < w(S)$, contradicting the assumption that (L, S, R) is a global minimum vertex-cut in G.

We conclude that the total number of edges in G connecting the vertices of L to the vertices of S is at least $|S|$, and so there must be some vertex $u \in L$ with $\deg_G(u) \geq \frac{|S|}{|L|} \geq n^{1-\epsilon}$, so $u \in T$ must hold. □

The following lemma provides a central tool for the proof of Theorem 3.5.

Lemma 3.7 *There is a randomized algorithm, that, given a simple n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$, and a vertex $x \in V(G)$, returns a vertex $y \in V(G)$, and the value c of the minimum x-y vertex-cut in G. Moreover, if there is a global minimum vertex-cut (L, S, R) in G with $x \in L$, then with probability at least $1 - 1/n^4$, $c = \mathsf{OPT}$ holds. The running time of the algorithm is $O\left(m^{1+o(1)} \cdot \log W'\right)$.*

We prove Lemma 3.7 below, after we complete the proof of Theorem 3.5 using it. Our algorithm starts by computing the set T of vertices of G, whose degrees in G are at least $n^{1-\epsilon}$. Clearly, T can be computed in time $O(m)$, and, as observed already, $|T| \leq 2n^{1-\epsilon}$ must hold. Next, we apply the algorithm from Lemma 3.7 to every vertex $x \in T$ in turn, and we let (c_x, y_x) be the outcome of this application of the algorithm from the lemma. We then let $x' \in T$ be the vertex for which $c_{x'}$ is minimized, and return $(c_{x'}, x', y_{x'})$ as the outcome of the algorithm. Lemma 3.7 guarantees that $c_{x'}$ is the value of the minimum x'-$y_{x'}$ vertex cut in G. Assume now that there is a global minimum vertex cut (L, S, R) in G with $|S| \geq n^{1-\epsilon} \cdot |L|$. Then, from Observation 3.6, $L \cap T \neq \emptyset$ must hold. Let v be any vertex in $L \cap T$. Then, when the algorithm from Lemma 3.7 was applied to vertex v, with probability at least $1 - 1/n^4$, the value c_v of the minimum v-y_v vertex cut that it returned is equal to OPT. In this case, we are guaranteed that the value c that our algorithm returns is equal to OPT as well. Since $|T| \leq n^{1-\epsilon}$, and since the running time of the algorithm from Lemma 3.7 is $O\left(m^{1+o(1)} \cdot \log W'\right)$, we get that the running time of our algorithm is $O\left(mn^{1-\epsilon+o(1)} \cdot \log W'\right)$.

In order to complete the proof of Theorem 3.5, it now remains to prove Lemma 3.7.

Proof of Lemma 3.7. Let $Z = V(G) \setminus (\{x\} \cup N_G(x))$. Let $\mathcal{A}''$ be an algorithm that selects a vertex $y \in Z$ at random, where the probability to select a vertex $v \in V(G)$ is $\frac{w(v)}{w(Z)}$. It then uses the algorithm from Corollary 2.6 to compute the value c of minimum x-y vertex cut in G, in time $O\left(m^{1+o(1)} \cdot \log W'\right)$, and then returns vertex y and the value c. We use the following observation.

Observation 3.8 *Suppose there is a global minimum vertex cut (L, S, R) in G with $x \in L$. Let (c, y) be the outcome of the algorithm $\mathcal{A}''$. Then with probability at least $\frac{1}{3}$, $y \in R$ and $c = \mathsf{OPT}$ holds.*

Proof: Consider the global minimum cut (L, S, R), and assume that $x \in L$. Let $S' = S \setminus N_G(x)$. Then from Claim 2.2, $w(S') \leq w(L)$ must hold.

We conclude that $w(Z) \leq w(L) + w(R) + w(S') \leq 2w(L) + w(R) \leq 3w(R)$, since we always assume by default that, in any vertex cut (L, S, R), $w(L) \leq w(R)$ holds. Therefore, the probability that Algorithm $\mathcal{A}''$ chooses a vertex $y \in R$ is $\frac{w(R)}{w(Z)} \geq \frac{1}{3}$. Clearly, if $y \in R$ holds, then the value c of the minimum x-y vertex cut in G is equal to OPT. □

We are now ready to complete the proof of Lemma 3.7. We execute Algorithm $\mathcal{A}''$ $\Theta(\log n)$ times, and output the smallest value c that the algorithm returned, together with the corresponding vertex y. It is immediate to verify that c is indeed the value of the minimum x-y vertex cut in G. Assume now that there is a global minimum vertex cut (L, S, R) in G with $x \in L$. We say that an application of Algorithm $\mathcal{A}''$ is *successful*, if it returns a value $c' = \mathsf{OPT}$. From Observation 3.8, the probability that a single application of the algorithm is successful is at least $1/3$. It is then easy to verify that the probability that every application of Algorithm $\mathcal{A}''$ was unsuccessful is at most $1/n^4$. If at least one application of the algorithm was successful, then we are guaranteed that the value c returned by our algorithm is equal to OPT. Since the running time of Algorithm $\mathcal{A}''$ is $O(m^{1+o(1)} \cdot \log W')$, the total running time of the entire algorithm is $O(m^{1+o(1)} \cdot \log W')$. □

3.3 Summary: an Algorithm for Non-Dense Graphs

By combining Algorithms Alg_1 and Alg_2, we obtain an algorithm for global minimum cut on graphs that are not very dense, that is summarized in the following corollary.

Corollary 3.9 *There is a randomized algorithm, whose input is a simple undirected n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$, and a parameter $0 < \epsilon \leq 1/2$, such that $m \leq n^{2-2\epsilon}$ holds. The algorithm returns two distinct vertices $x, y \in V(G)$ with $(x, y) \notin E(G)$, and a value c that is at least as large as the value of the minimum x-y vertex-cut in G. Moreover, with probability at least $1 - 1/n^4$, $c = \mathsf{OPT}$ holds. The running time of the algorithm is $O\left(mn^{1-\epsilon+o(1)} \cdot \log W'\right)$.*

Proof: Assume that we are given as input an undirected n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$, and a parameter $0 < \epsilon \leq \frac{1}{2}$, such that $m \leq n^{2-2\epsilon}$ holds. Consider the following algorithm, that we denote by Alg^*: the algorithm starts by applying Algorithm Alg_1 from Theorem 3.1 to G, and denotes by (c, x, y) its outcome. It then applies Algorithm Alg_2 from Theorem 3.5 to G, and denotes by (c', x', y') its outcome. If $c' \leq c$, it returns value $c^* = c'$, and vertices $x^* = x'$ and $y^* = y'$; otherwise, it returns value $c^* = c$ and vertices $x^* = x$ and $y^* = y$. Since the running times of both Algorithms Alg_1 and Alg_2 are bounded by $O(mn^{1-\epsilon+o(1)} \cdot \log W')$, we get that the running time of Algorithm $\mathcal{A}^*$ is $O(mn^{1-\epsilon+o(1)} \cdot \log W')$. It is easy to verify that the value of the minimum x^*-y^* cut in G is indeed at most c^*.

Consider now any global minimum cut (L, S, R) in G. Assume first that $|S| \geq n^{1-\epsilon} \cdot |L|$ holds. Then, from Theorem 3.5, with probability at least $1 - 1/n^4$, $c' = \mathsf{OPT}$ holds, and in this case, we are guaranteed that $c^* = \mathsf{OPT}$.

Assume now that $|S| < n^{1-\epsilon} \cdot |L|$. We claim that, in this case, there must be an integer $0 \leq i \leq \lceil \log W' \rceil$, such that $L_i \neq \emptyset$ and $|S_i| \leq |L_i| \cdot n^{1-\epsilon}$. Indeed, assume otherwise. Then for all $0 \leq i \leq \lceil \log W' \rceil$, $|S_i| \geq |L_i| \cdot n^{1-\epsilon}$ must hold, and so:

$$|S| = \sum_{i=0}^{\lceil \log W' \rceil} |S_i| \geq n^{1-\epsilon} \cdot \sum_{i=0}^{\lceil \log W' \rceil} |L_i| = |L| \cdot n^{1-\epsilon},$$

a contradiction. Theorem 3.1 then guarantees that, in this case, with probability at least $1 - 1/n^4$, $c = \mathsf{OPT}$ holds, and so we are guaranteed that $c^* = \mathsf{OPT}$.

In either case, we get that, with probability at least $1 - 1/n^4$, $c^* = \mathsf{OPT}$ holds. □

3.4 Algorithm Alg_3: when G is Dense and L is Small

Our final and the most involved algorithm, Alg_3, will only be used in the case where the input graph G is dense; we summarize it in the following theorem. The theorem statement uses the parameter $W_{\max} = W_{\max}(G)$, defined in Section 2.3.

Theorem 3.10 *There is a randomized algorithm, whose input is a simple undirected n-vertex graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$. The algorithm returns two distinct vertices $x, y \in V(G)$ with $(x, y) \notin E(G)$, and a value c that is at least as large as the value of the minimum x-y vertex-cut in G. Moreover, if there is a global minimum vertex-cut (L, S, R) in G, such that, for all $0 \leq i \leq \lceil \log W' \rceil$, $|S_i| \geq n^{44/45} \cdot |L_i|$ holds, then, with probability at least $\frac{1}{2^{11} \cdot \log^3 n \cdot \log W'}$, $c = \mathsf{OPT}$ holds. The running time of the algorithm is $O\left(n^{2.96} \cdot (\log(W_{\max}))^{O(1)}\right)$.*

The majority of the remainder of the paper is dedicated to the proof of Theorem 3.10. We now complete Theorem 1.1 using it.

3.5 Completing the Proof of Theorem 1.1

Recall that we are given as input simple undirected graph G with integral weights $1 \leq w(v) \leq W'$ on its vertices, where $W' \leq O(n^2 \cdot W)$. Our goal is to compute a global minimum vertex-cut in G, whose value is denoted by OPT. We start by presenting an algorithm with running time $O\left(mn^{0.99+o(1)} \cdot (\log W)^{O(1)}\right)$, followed by an algorithm with running time $O\left(m^{3/2+o(1)} \cdot (\log W)^{O(1)}\right)$.

Algorithm with Running Time $O\left(mn^{0.99+o(1)} \cdot (\log W)^{O(1)}\right)$

Let $\epsilon' = \frac{1}{100}$. We consider two cases. The first case happens if $m \leq n^{2-2\epsilon'} = n^{1.98}$ holds. In this case, we apply the algorithm from Corollary 3.9, and we denote by (c, x, y) its output. We then compute a minimum x-y vertex-cut (X, C, Y) in G using the algorithm from Corollary 2.6 in time $O(m^{1+o(1)} \cdot \log W')$, and return the cut (X, C, Y) as the algorithm's output. Notice that, from Corollary 3.9, with probability at least $1 - 1/n^4$, (X, C, Y) is indeed a global minimum vertex-cut in G. The running time of the algorithm from Corollary 3.9 is $O(mn^{1-\epsilon'+o(1)} \cdot \log W') = O(mn^{0.99+o(1)} \cdot \log W')$, and so the total running time of the algorithm in this case is $O(mn^{0.99+o(1)} \cdot \log W')$.

It now remains to consider the second case, where $m > n^{1.98}$ holds. We let $\epsilon = 1/45$. Consider the following algorithm, that we denote by Alg**: the algorithm starts by applying Algorithm Alg_1 from Theorem 3.1 to G, and denotes by (c_1, x_1, y_1) its outcome. It then performs $\hat{z} = \left\lceil 2^{18} \cdot \log W' \cdot \log^4 n \right\rceil$ iterations, and in every iteration it applies Algorithm Alg_3 from Theorem 3.10 to G. We denote by (c_3, x_3, y_3) the outcome of the iteration in which the value c returned by the algorithm was the smallest. If $c_3 \leq c_1$, then we use the algorithm from Corollary 2.6 to compute a minimum x_3-y_3 vertex-cut (X_3, C_3, Y_3) in G, and return this cut as the algorithm's outcome. Otherwise, we use the algorithm from Corollary 2.6 to compute a minimum x_1-y_1 vertex-cut (X_1, C_1, Y_1), and return this cut.

Recall that the running time of Algorithms Alg_1 from Theorem 3.1 is $O\left(mn^{1-\epsilon+o(1)} \cdot \log W'\right) \leq O(mn^{44/45+o(1)} \cdot \log W')$, and the running time of Algorithm Alg_3 from Theorem 3.10 is:

$$O\left(n^{2.96} \cdot (\log(W_{\max}))^{O(1)}\right) \leq O\left(n^{2.96+o(1)} \cdot (\log W')^{O(1)}\right),$$

since $W_{\max} \leq O(n \cdot W')$ from Inequality 1.

Since the running time of the algorithm from Corollary 2.6 is $O(m^{1+o(1)} \cdot \log W')$, we get that the total running time of algorithm $\mathcal{A}^{**}$ is bounded by:

$$\begin{aligned}
&\hat{z} \cdot O\left(n^{2.96+o(1)} \cdot (\log W')^{O(1)}\right) + O(mn^{44/45+o(1)} \cdot \log W') \\
&\qquad \leq O\left(n^{2.96+o(1)} \cdot (\log W')^{O(1)}\right) + O(mn^{44/45+o(1)} \cdot \log W') \\
&\qquad \leq O\left(mn^{0.99+o(1)} \cdot (\log W')^{O(1)}\right) + O(mn^{44/45+o(1)} \cdot \log W') \\
&\qquad \leq O\left(mn^{0.99+o(1)} \cdot (\log W')^{O(1)}\right).
\end{aligned}$$

since $\hat{z} = O\left(\log W' \cdot \log^4 n\right)$ and $m > n^{1.98}$.

Consider now any global minimum vertex-cut (L, S, R) in G. Assume first that, for some integer $0 \leq i \leq \lceil \log W' \rceil$, $L_i \neq \emptyset$ and $|S_i| \leq |L_i| \cdot n^{44/45} = |L_i| \cdot n^{1-\epsilon}$ hold. From Theorem 3.1, in this case, with probability at least $1 - 1/n^4$, $c_1 = \mathsf{OPT}$ holds, and so our algorithm returns a vertex cut of value OPT.

Otherwise, we are guaranteed that for all $0 \leq i \leq \lceil \log W' \rceil$, $|S_i| \geq n^{44/45} \cdot |L_i|$ holds. We say that the application of the algorithm from Theorem 3.10 is successful, if the estimate c that it returns is equal to OPT. From Theorem 3.10, the probability that a single application of the algorithm is successful is at least $\frac{1}{2^{11} \cdot \log^3 n \cdot \log W'}$. Since we execute the algorithm from Theorem 3.10 $\hat{z} = \left\lceil 2^{18} \log W' \cdot \log^4 n \right\rceil$ times, the probability that every application of the theorem is unsuccessful is at most $\left(1 - \frac{1}{2^{11} \cdot \log^3 n \cdot \log W'}\right)^{\hat{z}} \leq \frac{1}{n}$. If at least one application of the algorithm is successful, then our algorithm is guaranteed to return a vertex cut of value OPT. Note that in either case, our algorithm returns global a minimum vertex-cut with probability at least $1 - 1/n$.

The total running time of the algorithm is:

$$O\left(mn^{0.99+o(1)} \cdot \log W'\right) \leq O\left(mn^{0.99+o(1)} \cdot \log W\right),$$

since $W' = O(n^2 \cdot W)$.

Algorithm with Running Time $O\left(m^{3/2+o(1)} \cdot (\log W)^{O(1)}\right)$

Note that we can assume that the input graph G is connected, and that it is not a tree, since otherwise the problem is trivial. We can also assume that $m < n^2$, since otherwise, the $\tilde{O}(mn)$-time algorithm of [HRG00] achieves the desired bound of $O\left(m^{3/2+o(1)} \cdot (\log W)^{O(1)}\right)$ on the running time. Therefore, we can denote $m = n^x$, where $1 \leq x < 2$. Let $\epsilon = 1 - x/2$, so that $m = n^{2-2\epsilon}$ and $\epsilon > 0$. We apply the algorithm from Corollary 3.9 to graph G, which must return a global minimum vertex cut with probability at least $1 - 1/n^4$. The running time of the algorithm is $O\left(mn^{1-\varepsilon+o(1)} \cdot (\log W')^{O(1)}\right) \leq O\left(m^{3/2+o(1)} \cdot (\log W)^{O(1)}\right)$.

In order to complete the proof of Theorem 1.1, it is now enough to prove Theorem 3.10, which we do in the remainder of the paper. We start with a high-level overview of the proof.

3.6 Proof of Theorem 3.10 – High Level Overview

Throughout the proof, we will use the following simple claim.

Claim 3.11 *Let G be a simple undirected n-vertex graph with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$, and let $0 < \epsilon < 1$ be a parameter. Let (L, S, R) be any vertex cut in G, and assume that, for all $0 \leq i \leq \lceil \log W' \rceil$, $|S_i| \geq n^{1-\epsilon} \cdot |L_i|$ holds. Then $|L| \leq \frac{|S|}{n^{1-\epsilon}} \leq n^{\epsilon}$ holds.*

Proof: Since, for all $0 \leq i \leq \lceil \log W' \rceil$, $|S_i| \geq n^{1-\epsilon} \cdot |L_i|$ holds, we get that:

$$|S| = \sum_{i=0}^{\lceil \log W' \rceil} |S_i| \geq \sum_{i=0}^{\lceil \log W' \rceil} |L_i| \cdot n^{1-\epsilon} \geq |L| \cdot n^{1-\epsilon}.$$

Therefore, $|L| \leq \frac{|S|}{n^{1-\epsilon}} \leq n^{\epsilon}$. □

We assume that we are given a simple undirected n-vertex graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$. Throughout, we set $\epsilon = 1/45$. We say that a global minimum vertex-cut (L, S, R) is *acceptable*, if, for all $0 \leq i \leq \lceil \log W' \rceil$, $|S_i| \geq n^{1-\epsilon} \cdot |L_i|$ holds. We define a specific global minimum vertex-cut (L, S, R) as follows: if an acceptable global minimum vertex-cut exists in G, then we let (L, S, R) be any such cut; otherwise, we let (L, S, R) be an arbitrary vertex cut. The cut (L, S, R) remains fixed throughout this section. We denote by $\mathsf{OPT} = w(S)$ the value of the global minimum vertex-cut.

3.6.1 A Good Set of Terminals and a Good Pair

We use the notion of a good set of terminals, that is defined next.

Definition 3.12 (A good set of terminals) *Let $T \subseteq V(G)$ be any subset of vertices. We say that T is a* good set of terminals *with respect to a global minimum cut (L, S, R) if $|T \cap L| = 1$ and $T \cap R \neq \emptyset$ hold, and, additionally, if we denote by x the unique vertex in $T \cap L$, then $T \cap S \subseteq N_G(x)$.*

Since we have fixed the global minimum cut (L, S, R), if vertex set T is a good set of terminals with respect to this cut, then we simply say that T is a good set of terminals.

Next, we provide a randomized algorithm, that we refer to as AlgSelectTerminals, that selects a subset T of vertices of G so that, if the cut (L, S, R) is acceptable, then, with a high enough probability, set T of terminals is good.

The notion of a good set of terminals was implicitly introduced by [LNP+21] in their algorithm for computing a global minimum vertex-cut in unweighted graphs. They also provide a simple randomized algorithm that, with a high enough probability, computes a good set of terminals in this setting. Our algorithm AlgSelectTerminals is a natural extension of their algorithm to vertex-weighted graphs.

We now describe Algorithm AlgSelectTerminals formally. The algorithm starts by selecting an integer $i \in [1, \lceil \log(n \cdot W') \rceil]$ uniformly at random. Next, we let T be a random subset of vertices of G, obtained by adding every vertex $v \in V(G)$ to T independently with probability $\frac{w(v)}{2^{i+1}}$. We say that a good event $\mathcal{E}$ happens if the resulting set T of vertices is a good set of terminals with respect to (L, S, R). If the event $\mathcal{E}$ happens, then we say that Algorithm AlgSelectTerminals is *successful*. In the following claim, we show that, if the cut (L, S, R) is acceptable, then algorithm AlgSelectTerminals is successful with a reasonably high probability.

Claim 3.13 *If the cut (L, S, R) is acceptable, then $\mathbf{Pr}[\mathcal{E}] \geq \frac{1}{2^{11} \log n \cdot \log W'}$.*

Proof: Assume that the cut (L, S, R) is acceptable. Let $1 \leq i^* \leq \lceil \log(n \cdot W') \rceil$ be the unique integer for which $2^{i^*-1} \leq w(L) < 2^{i^*}$ holds. We say that a good event $\mathcal{E}'$ happens if $i = i^*$. Clearly, $\mathbf{Pr}[\mathcal{E}'] \geq \frac{1}{\lceil \log(n \cdot W') \rceil} \geq \frac{1}{2 \log n \cdot \log W'}$.

For every vertex $v \in L$, we say that a good event $\mathcal{E}(v)$ happens, if all of the following hold: (i) $v \in T$; (ii) v is the unique vertex of L that lies in T; and (iii) $T \cap S \subseteq N_G(v)$. Let $S'_v = S \setminus N_G(v)$, and observe that from Claim 2.2, $w(S'_v) \leq w(L)$ must hold. Let $X = (L \cup S'_v) \setminus \{v\}$. Then Event $\mathcal{E}(v)$ happens if and only if v is added to T, but none of the vertices of X are added to T. From our discussion so far, $w(X) \leq 2w(L)$, and for every vertex $x \in X$, $w(x) \leq w(L) \leq 2^{i^*}$. Altogether, we get that:

$$\begin{aligned}
\mathbf{Pr}\left[\mathcal{E}(v) \mid \mathcal{E}'\right] &= \frac{w(v)}{2^{i+1}} \cdot \prod_{x \in X} \left(1 - \frac{w(x)}{2^{i+1}}\right) \\
&= \frac{w(v)}{2^{i+1}} \cdot e^{\sum_{x \in X} \ln\left(1 - \frac{w(x)}{2^{i+1}}\right)} \\
&\geq \frac{w(v)}{16 \cdot 2^i}.
\end{aligned}$$

For the last inequlatiy, we used the fact that, from Taylor's expansion of function $\ln(1-\epsilon)$, for all $0 \leq \epsilon \leq 1/2$, $\ln(1-\epsilon) > -\epsilon - \epsilon^2 \geq -2\epsilon$ holds. Therefore, for all $x \in X$, $\ln\left(1 - \frac{w(x)}{2^{i+1}}\right) \geq -\frac{w(x)}{2^i}$, and $\sum_{x \in X} \ln\left(1 - \frac{w(x)}{2^{i+1}}\right) \geq -\frac{\sum_{x \in X} w(x)}{2^i} = -\frac{w(X)}{2^i} \geq -2$, since $w(X) \leq 2w(L) \leq 2^{i+1}$.

Let $\mathcal{E}_1$ be the good event that Event $\mathcal{E}(v)$ happened for any vertex $v \in L$. Notice that the events $\mathcal{E}(v)$ are disjoint for vertices $v \in L$. Therefore:

$$\mathbf{Pr}\left[\mathcal{E}_1 \mid \mathcal{E}'\right] = \sum_{v \in L} \mathbf{Pr}\left[\mathcal{E}(v) \mid \mathcal{E}'\right] \geq \frac{w(L)}{16 \cdot 2^i} \geq \frac{1}{32},$$

since $w(L) \geq 2^{i-1}$. Next, we let $\mathcal{E}_2$ be the good event that $R \cap T \neq \emptyset$. Since we assumed that $w(R) \geq w(L)$, it is immediate to verify that:

$$\mathbf{Pr}\left[\mathcal{E}_2 \mid \mathcal{E}'\right] \geq \mathbf{Pr}\left[T \cap L \neq \emptyset \mid \mathcal{E}'\right] \geq \mathbf{Pr}\left[\mathcal{E}_1 \mid \mathcal{E}'\right] \geq \frac{1}{32}.$$

Since Events $\mathcal{E}_1$ and $\mathcal{E}_2$ are independent, we get that:

$$\mathbf{Pr}\left[\mathcal{E}_1 \wedge \mathcal{E}_2 \mid \mathcal{E}'\right] = \mathbf{Pr}\left[\mathcal{E}_1 \mid \mathcal{E}'\right] \cdot \mathbf{Pr}\left[\mathcal{E}_2 \mid \mathcal{E}'\right] \geq \frac{1}{2^{10}}.$$

Finally, we get that:

$$\mathbf{Pr}\left[\mathcal{E}\right] \geq \mathbf{Pr}\left[\mathcal{E}_1 \wedge \mathcal{E}_2 \mid \mathcal{E}'\right] \cdot \mathbf{Pr}\left[\mathcal{E}'\right] \geq \frac{1}{2^{11} \cdot \log n \cdot \log W'}.$$

□

Given a set T of vertices of G, and a vertex $x \in T$, we denote by $T_x = T \setminus (\{x\} \cup N_G(x))$. Next, we define a notion of a good pair.

Definition 3.14 (A good pair) *Let $T \subseteq V(G)$ be a set of vertices, let $x \in T$ be a vertex, and let (L, S, R) be a global minimum vertex-cut. We say that (x, T) is a* good pair *with respect to the cut (L, S, R), if T is a good set of terminals with respect to (L, S, R), and x is the unique vertex of $T \cap L$.*

Notice that, if (x, T) is a good pair with respect to (L, S, R), then $T \cap S \subseteq N_G(x)$ and $T \cap R \neq \emptyset$ must hold; in particular, $T_x \subseteq R$. Therefore, the value of the minimum x-T_x vertex-cut in G is OPT (since the set S of vertices disconnects x from T_x in G, and S is disjoint from x and from T_x). Clearly, if T is a good set of terminals with respect to (L, S, R), then there is exactly one vertex $x \in T$, such that (x, T) is a good pair with respect to (L, S, R).

The main technical subroutine of our algorithm, that we formally describe below, receives as input a set T of vertices of G, and a vertex $x \in T$. The subroutine outputs a value c_x that is at least as high as the value of the minimum x-T_x cut in G. If, additionally, (L, S, R) is an acceptable cut, and (x, T) is a good pair with respect to (L, S, R), then the subroutine guarantees that, with a reasonably high probability, c_x is equal to the value of the minimum x-T_x vertex-cut in G, and hence to OPT. One subtlety in the description of the subroutine is that it needs an access to a *subgraph oracle*, that we define below. We will ensure that the subroutine only accesses the oracle $O(\log n \log(W_{\max}))$ times over its execution. We will first provide the statement of the theorem summarizing the subroutine, assuming that it has access to the subgraph oracle, and then show an algorithm that, in a sense, implements this oracle. We now proceed to define a subgraph oracle.

3.6.2 A Subgraph Oracle

We now define a subgraph oracle, that receives as input an undirected graph G with integral weights on its vertices. The definition uses the parameter $W_{\max} = W_{\max}(G)$ defined in Section 2.2.

Definition 3.15 (A Subgraph Oracle) *A δ-subgraph oracle, for a parameter $0 < \delta \leq 1/2$, receives as input a simple undirected n-vertex graph G with integral weights $1 \leq w(v) \leq W'$ on its vertices $v \in V(G)$ in the adjacency list representation, where n is sufficiently large, so $\delta \geq \frac{1}{\sqrt{\log n}}$, and a set $Z \subseteq V(G)$ of vertices. The oracle must return a partition (Y^h, Y^ℓ) of $V(G)$, and a collection E' of at most $n^{2-\delta} \cdot \log^2(W_{\max})$ edges of G, such that $E' = E_G(Y^\ell, Z)$ holds. We say that the oracle* errs, *if some vertex $v \in Y^h$ has fewer than $\frac{n^{1-\delta}}{1000 \log n}$ neighbors in Z, or some vertex $v' \in Y^\ell$ has more than $n^{1-\delta} \cdot \log^2(W_{\max})$ such neighbors. We require that the probability that the oracle errs is at most $\frac{1}{n^5 \cdot \log^4(W_{\max})}$.*

3.6.3 The Main Technical Subroutine

The following theorem summarizes our main technical subroutine. The theorem also uses the notion of the split graph and the parameter $W_{\max} = W_{\max}(G)$, both of which are defined in Section 2.3.

Theorem 3.16 *There is a randomized algorithm, whose input consists of a simple undirected n-vertex graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$ in the adjacency-list representation, a subset $T \subseteq V(G)$ of its vertices, and a vertex $s \in T$. Additionally, the algorithm is given two copies of the split graph G' corresponding to G in the modified adjacency-list representation. Lastly, it is given an access to the δ-subgraph oracle for G, for $\delta = \frac{4}{45} - \frac{\log(4000 \log n)}{\log n} = \frac{4}{45} - o(1)$. The algorithm returns a value c_s, that is at least as high as the value of the minimum s-T_s vertex-cut in G. Moreover, if there is a global minimum vertex-cut (L, S, R) in G, such that $|S| \geq n^{44/45} \cdot |L|$ holds, and (s, T) is a good pair with respect to cut (L, S, R), then, with probability at least $\frac{1}{2}$, c_s is equal to the value of the minimum s-T_s vertex-cut in G. The running time of the algorithm is $O\left(n^{86/45+o(1)} \cdot (\log(W_{\max}))^{O(1)}\right)$, and it accesses the δ-subgraph oracle at most $8 \log n \cdot \log(W_{\max})$ times.*

We note that the running time of the algorithm from Theorem 3.16 may be faster than $|E(G)|$, and that it might modify the adjacency-list representations of the graph G' that it is given as input. Specifically, the algorithm gradually modifies the graph G' in order to "simplify" it, and also maintains an s-T_s flow f in the resulting modified graph, that it gradually augments. One copy of the graph will be used to maintain the modified graph G', while the second copy will be used to maintain the residual flow network of G' with respect to the flow that the algorithm maintains. We prove Theorem 3.16 in the remainder of the paper, after we complete the proof of Theorem 3.10 using it. We start by providing an algorithm that we will use to implement the oracle.

3.6.4 Implementing the Oracle

Recall that our algorithm will apply the subroutine from Theorem 3.16 to every vertex $s \in T$. Each such execution of the subroutine will make at most $8 \log n \cdot \log(W_{\max})$ calls to the oracle. In order to make our algorithm efficient, we will design an oracle that responds to several δ-subgraph queries in bulk. Intuitively, for all $1 \leq i \leq \lfloor 8 \log n \cdot \log(W_{\max}) \rfloor$, we will run the subroutine from Theorem 3.16 for each vertex $s \in T$, until its ith attempt to access the oracle. We will then simultaneously compute responses to all resulting $|T|$ subgraph queries. We summarize our algorithm for implementing the oracle in the following lemma. The lemma statement uses the matrix multiplication exponent ω, whose current bound is $\omega \leq 2.371552$ [WXXZ24].

Lemma 3.17 *There is a deterministic algorithm, that is given as input an undirected n-vertex graph G with weights $1 \leq w(v) \leq W'$ on the vertices $v \in V(G)$, a parameter $\frac{1}{\sqrt{\log n}} < \delta \leq \frac{1}{2}$, and $q \leq n$ queries $(Z_1, \ldots, Z_q)$ to the δ-subgraph oracle. The algorithm computes responses $(Y_1^h, Y_1^\ell, E_1'), \ldots, (Y_q^h, Y_q^\ell, E_q')$ to the queries, such that the probability that the algorithm errs in its response to any query is at most $\frac{1}{n^9 \cdot \log^4(W_{\max})}$. The running time of the algorithm is $O\left(n^{3-\delta \cdot 2(3-\omega)/(5-\omega)+o(1)} \cdot \operatorname{poly}\log(W_{\max})\right) \leq O\left(n^{3-0.478\delta+o(1)} \cdot \operatorname{poly}\log(W_{\max})\right)$, where ω is the matrix multiplication exponent.*

Proof: We start by computing, for all $1 \leq i \leq q$, the partition (Y_i^h, Y_i^ℓ) of $V(G)$, as follows. Fix an index $1 \leq i \leq q$, and set $\tau = \frac{n^{1-\delta}}{1000 \log n}$. We apply the algorithm for the Undirected Degree Estimation problem from Claim 2.11 to graph G, the sets $Z = Z_i$ and $Z' = V(G)$ of its vertices, and threhsold τ. Let $A \subseteq V(G)$ be the subset of vertices that the algorithm returns. We then set $Y_i^h = A$ and $Y_i^\ell = V(G) \setminus A$. Let $\mathcal{E}_i$ be the bad event that the algorithm from Claim 2.11 erred, and recall that

$\mathbf{Pr}\left[\mathcal{E}_i\right] \leq \frac{1}{n^{10}\log^4(W_{\max})}$. Note that, if the event $\mathcal{E}_i$ did not happen, then we are guaranteed that (i) every vertex $v \in Y_i^h$ has at least $\tau = \frac{n^{1-\delta}}{1000\log n}$ neigbhors in Z_i; and (ii) every vertex $v' \in Y_i^\ell$ has at most $1000\tau \log n \log(W_{\max}) = n^{1-\delta} \cdot \log(W_{\max})$ neighbors in Z. Recall that the running time of the algorithm from Claim 2.11 is:

$$O\left(\frac{n \cdot |Z_i| \cdot \log n \cdot \log(W_{\max})}{\tau}\right) \leq O\left(n^{1+\delta} \cdot \log^2 n \cdot \log(W_{\max})\right) \leq O\left(n^{2-\delta} \cdot \log^2 n \cdot \log(W_{\max})\right),$$

since $\delta \leq 1/2$, and so the time required to execute this step for all $1 \leq i \leq q$ is $O\left(n^{3-\delta} \cdot \log^2 n \cdot \log(W_{\max})\right)$. We let $\mathcal{E}$ be the bad event that $\mathcal{E}_i$ happened for any $1 \leq i \leq q$. By using the Union Bound, and since $q \leq n$, we get that $\mathbf{Pr}\left[\mathcal{E}\right] \leq \frac{1}{n^9\log^4(W_{\max})}$.

Next, we construct a tripartite graph $H = (A \cup B \cup C, E')$, as follows. We let $A = \{a_v \mid v \in V(G)\}$, $B = \{b_v \mid v \in V(G)\}$, and $C = \{c_1, \ldots, c_q\}$. For every edge $e = (u, v) \in E(G)$, we add the edges (a_v, b_u) and (a_u, b_v) to H. Additionally, for every index $1 \leq i \leq q$, for every vertex $v \in Z_i$, we add the edge (a_v, c_i), and for every vertex $u \in Y_i^\ell$, we add the edge (c_i, b_u) to H.

For a triple $\Pi = (a_v, b_u, c_i)$ of vertices of H with $a_v \in A$, $b_u \in B$, and $c_i \in C$, we say that Π is a *good triple*, if $(v, u) \in E_G(Z_i, Y_i^\ell)$. Notice that, if Event $\mathcal{E}$ did not happen, then $|E_G(Z_i, Y_i^\ell)| \leq n^{2-\delta} \cdot \log(W_{\max})$ must hold. Therefore, if Event $\mathcal{E}$ did not happen, then the total number of good triples in H is bounded by $n^{2-\delta} \cdot q \cdot \log(W_{\max}) \leq n^{3-\delta} \cdot \log(W_{\max})$. Moreover, every good triple defines a triangle in H and vice versa. Therefore, in order to compute, for every index $1 \leq i \leq q$, the set $E_G(Z_i, Y_i^\ell)$ of edges, it is sufficient to compute all triangles in H. We use the following algorithm of [BPWZ14] in order to do so.

Theorem 3.18 (Theorem 1 in [BPWZ14]) *There is a deterministic algorithm that lists all triangles in an n-vertex graph in time:*

$$O\left(n^{\omega+o(1)} + n^{3(\omega-1)/(5-\omega)+o(1)} \cdot t^{2(3-\omega)/(5-\omega)}\right),$$

where ω is the matrix multiplication exponent, and t is the total number of triangles in G.

By substituting $t \leq n^{3-\delta} \cdot \log(W_{\max})$, we get that the running time of the algorithm from Theorem 3.18 on graph H is bounded by:

$$\begin{aligned} O\left(\left(n^\omega + n^{3(\omega-1)/(5-\omega)} \cdot n^{2(3-\omega)\cdot(3-\delta)/(5-\omega)}\right) \cdot n^{o(1)} \cdot \mathrm{poly}\log(W_{\max})\right) \\ \leq O\left(n^{3-\delta\cdot 2(3-\omega)/(5-\omega)}\right) \cdot n^{o(1)} \cdot \mathrm{poly}\log(W_{\max}). \end{aligned}$$

Finally, recall that the running time of the first step, that performed q applications of the algorithm from Claim 2.11 for the Degree Estimation problem is $O(n^{3-\delta} \cdot \log^2 n \cdot \log(W_{\max}))$, and that the time required to compute the graph H, given the graph G and the sets $\left\{Z_i, Y_i^\ell\right\}_{1\leq i\leq q}$ of vertices is bounded by $O(n^2)$. Therefore, the total running time of the algorithm is $O\left(n^{3-\delta\cdot 2(3-\omega)/(5-\omega)} \cdot n^{o(1)} \cdot \mathrm{poly}\log(W_{\max})\right)$, or $O\left(n^{3-0.478\delta+o(1)} \cdot \mathrm{poly}\log(W_{\max})\right)$ using the current bound $\omega \leq 2.371552$ [WXXZ24]. □

3.6.5 Completing the Proof of Theorem 3.10

We are now ready to complete the proof of Theorem 3.10. We assume that we are given as input a simple undirected n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$. We fix a global minimum vertex-cut (L, S, R) in G as follows. If there exists an acceptable global minimum vertex cut (L, S, R) (that is, for all $0 \leq i \leq \lceil \log W' \rceil$, $|S_i| \geq n^{44/45} \cdot |L_i|$ holds), then we fix this cut. Otherwise, we let (L, S, R) be an arbitrary global minimum vertex-cut.

We start by computing a split graph G' of G, and two copies of G' in modified the adjacency-list representation. We also compute a matrix representation of G. All this can be done in time $O(m \log(W_{\max})) \leq O(n^2 \log(W_{\max}))$. Next, we apply algorithm AlgSelectTerminals to compute a set $T \subseteq V(G)$ of vertices that we call terminals from now on. It is easy to verify that the running time of Algorithm AlgSelectTerminals is $O(n)$. We let $\mathcal{E}_T$ be the good event that T is a good set of terminals with respect to the cut (L, S, R). From Claim 3.13, if (L, S, R) is an acceptable cut, then $\mathbf{Pr}[\mathcal{E}_T] \geq \frac{1}{2^{11} \cdot \log n \cdot \log W'}$.

Our algorithm consists of at most $z = \lfloor 8 \log n \cdot \log W_{\max} \rfloor$ phases. For all $1 \leq i \leq z$, Phase i has $|T|$ iterations, each of which processes a different vertex $s \in T$. When vertex $s \in T$ is processed in the ith phase, we apply the algorithm from Theorem 3.16 to graph G, set T of terminals and vertex s. We supply the algorithm with the 2 copies of the adjacency-list representation of G'. We execute the algorithm until its ith call to the δ-subgraph oracle, and we denote by $Z_{i,s}$ the resulting query to the oracle. We also record all random choices made by the algorithm, and, when the algorithm from Theorem 3.16 is executed with vertex s as an input in Phase $(i+1)$, we repeat all the recorded random choices, so that the first i calls to the δ-subgraph oracle are identical to the execution of the algorithm in Phase i. Lastly, once the algorithm from Theorem 3.16 performs the ith call to the δ-subgraph oracle, we terminate it, and we undo all the modifications that it made to the two copies of graph G' in the modified adjacency-list representation. We then continue to process the next vertex of T.

Once all vertices of T are processed, we execute the algorithm from Lemma 3.17 in order to compute responses to all queries $\{Z_{i,s} \mid s \in T\}$. We then continue to Phase $(i+1)$. Note that, at the beginning of Phase $(i+1)$, for all $s \in T$, we have computed the responses to the first i queries to the δ-subgraph oracle that the algorithm from Theorem 3.16 asks when applied to s.

Let $\mathcal{E}'_T$ be the bad event that the algorithm from Lemma 3.17 errs in any of the iterations. Then, from Lemma 3.17 and the Union Bound, $\mathbf{Pr}[\mathcal{E}'_T] \leq \frac{8 \log n \cdot \log(W_{\max})}{n^9 \cdot \log^4(W_{\max})} \leq \frac{8}{n^9 \log^2(W_{\max})}$, since $W_{\max} \geq n$ by definition.

After the last phase terminates, we obtain, for every vertex $s \in T$, the value c_s that the algorithm from Theorem 3.16 outputs. Recall that c_s is at least as high as the value of the minimum s-T_s vertex cut in G. We let $x \in T$ be the vertex minimizing the value c_x, and we let y be any vertex of T_x. We compute the minimum x-y vertex cut in G using the algorithm from Corollary 2.6, whose running time is $O\left(m^{1+o(1)} \cdot \log(W_{\max})\right) \leq O\left(n^{2+o(1)} \cdot \log(W_{\max})\right)$, and return the value c^*_x of the cut, and the vertices x and y. Clearly, $c^*_x \geq \mathsf{OPT}$ must hold.

Assume now that the cut (L, S, R) is acceptable, so for all $0 \leq i \leq \lceil \log W' \rceil$, $|S_i| \geq n^{44/45} \cdot |L_i|$ holds. Assume further that Event $\mathcal{E}_T$ happened. Let s be the unique vertex in $T \cap L$, and let $\mathcal{E}''_s$ be the good event that the algorithm from Theorem 3.16, when applied to s, returned a value c_s that is equal to the value of the minimum s-T_s vertex cut in G. Then, from Theorem 3.16, $\mathbf{Pr}[\mathcal{E}''_s \mid \mathcal{E}_T \wedge \neg \mathcal{E}'_T] \geq \frac{1}{2}$. From our discussion, if Events $\mathcal{E}_T$ and $\mathcal{E}''_s$ happen, and Event $\mathcal{E}'_T$ does not happen, then the algorithm is guaranteed to return an estimate $c^*_x = \mathsf{OPT}$. Therefore, the probability that $c^*_x = \mathsf{OPT}$ is at least:

$$\begin{aligned}\mathbf{Pr}\left[\mathcal{E}_T \wedge \mathcal{E}''_s \wedge \neg\mathcal{E}'_T\right] &= \mathbf{Pr}\left[\mathcal{E}''_s \mid \mathcal{E}_T \wedge \neg\mathcal{E}'_T\right] \cdot \mathbf{Pr}\left[\neg\mathcal{E}'_T \mid \mathcal{E}_T\right] \cdot \mathbf{Pr}\left[\mathcal{E}_T\right] \\ &= \mathbf{Pr}\left[\mathcal{E}''_s \mid \mathcal{E}_T \wedge \neg\mathcal{E}'_T\right] \cdot \mathbf{Pr}\left[\neg\mathcal{E}'_T\right] \cdot \mathbf{Pr}\left[\mathcal{E}_T\right] \\ &\geq \frac{1}{2} \cdot \left(1 - \frac{8}{n^9 \cdot \log^2(W_{\max})}\right) \cdot \frac{1}{2^{11} \cdot \log n \cdot \log(W')} \\ &\geq \frac{1}{2^{13} \cdot \log n \cdot \log(W')}.\end{aligned}$$

We now bound the running time of the algorithm. Recall that the running time of the algorithm from Theorem 3.16 is $O\left(n^{86/45+o(1)} \cdot (\log(W_{\max}))^{O(1)}\right)$. We execute this algorithm for $|T| \leq n$ terminals, and for each terminal $s \in T$, we excute the algorithm $O(\log n \cdot \log W_{\max})$ times (though some of these executions are incomplete). Therefore, the total time that we spend on executing the algorithm from Theorem 3.16 is $O\left(n^{3-4/45+o(1)} \cdot (\log(W_{\max}))^{O(1)}\right)$. Additionally, we perform $O(\log n \cdot \log W_{\max})$ calls to the algorithm from Lemma 3.17 for the δ-subgraph oracle, where $\delta = \frac{4}{45} - o(1)$. From Lemma 3.17, the total running time of all such calls is bounded by:

$$O\left(n^{3-0.478\delta+o(1)} \cdot \operatorname{poly}\log(W_{\max})\right) \leq O\left(n^{2.96} \cdot (\log(W_{\max}))^{O(1)}\right).$$

The running time of Algorithm AlgSelectTerminals is bounded by $O(n \log(W_{\max}))$, and so the total running time of the entire algorithm is bounded by $O\left(n^{2.96} \cdot (\log(W_{\max}))^{O(1)}\right)$.

In order to complete the proof of Theorem 3.10 and of Theorem 1.1, it is now enough to prove Theorem 3.16, which we do in the remainder of the paper.

4 Main Subroutine: the Proof of Theorem 3.16

We assume that we are given as input a simple undirected n-vertex graph G with integral weights $1 \leq w(v) \leq W'$ on vertices $v \in V(G)$, a subset $T \subseteq V(G)$ of vertices that we call terminals, and a vertex $s \in T$. Additionally, the algorithm is given two copies of the split graph G' corresponding to G in the modified adjacency-list representation. For every edge $e \in E(G')$, its capacity in G' is denoted by $c(e)$. Recall that we denoted by $W'_{\max}$ the smallest integral power of 2 with $W'_{\max} > \max_{v \in V(G)}\{w(v)\}$, and by $W_{\max}$ the smallest integral power of 2 with $W_{\max} \geq 2n \cdot W'_{\max}$. For every edge $e \in E(G')$, if e is a special edge then $c(e) < W'_{\max}$, and if it is a regular edge, then $c(e) = W_{\max}$ hold.

For the sake of the analysis, we fix a minimum vertex-cut (L, S, R) in G, as follows. If there exists a minimum vertex-cut (L', S', R') with $|S'| \geq n^{44/45} \cdot |L'|$, such that (s, T) is a good pair with respect to (L', S', R'), then we set $(L, S, R) = (L', S', R')$. Otherwise, we let (L, S, R) be any global minimum vertex-cut in G. For brevity, if (s, T) is a good pair with respect to this fixed cut (L, S, R), we will simply say that (s, T) is a *good pair*; in particular, $|S| \geq n^{44/45} \cdot |L|$ must hold in this case. Recall that we have denoted by $T_s = T \setminus (\{s\} \cup N_G(s))$, and, if (s, T) is a good pair, then $T_s \subseteq R$ holds.

We start by modifying the graph G' as follows. We add a new destination vertex t to it, and, for every vertex $v \in T_s$, we add the edge (v^{in}, t) of capacity $W_{\max}$ to the graph. These newly added edges are considered regular edges. Notice that this modification can be done in time $O(n)$, and that the value of the minimum s^{out}-t edge-cut in G' is equal to the value of the minimum s-T_s vertex-cut in G. Moreover, if (s, T) is a good pair, then the set $E_S = \left\{(v^{\text{in}}, v^{\text{out}}) \mid v \in S\right\}$ of edges of G' is the minimum s^{out}-t edge-cut in G', and $c(E_S) = \mathsf{OPT}$, the value of the global minimum vertex-cut in G.

We denote by OPT_s the value of the minimum s^{out}-t edge-cut in this modified graph G'. Our goal now is to output a value $c_s \geq \mathsf{OPT}_s$. Additionally, if (s,T) is a good pair, then we need to ensure that, with probability at least $\frac{1}{2}$, $c_s = \mathsf{OPT}_s$ holds.

Over the course of the algorithm, we may modify the graph G' by performing *shortcut operations*, that we define next.

Shortcut Operations. In a shortcut operation, we add an edge (v,t) of capacity $W_{\max}$ to graph G', for some vertex $v \in V(G')$. All such newly added edges are considered regular edges. If (s,T) is a good pair, then the shortcut operation in which an edge (v,t) is inserted into G' is a *valid shortcut operation*, and edge (v,t) is a *valid shortcut edge*, if and only if either (i) v is a copy of a vertex $u \in R$; or (ii) v is an out-copy of a vertex $u' \in S$. If the pair (s,T) is not good, then any such shortcut operation is a valid shortcut operation, and any edge (v,t) is a valid shortcut edge. We will use the following claim that summarizes the properties of the graph obtained from G' via a sequence of shortcut operations.

Claim 4.1 *Let G'' be a graph that is obtained from G' via a sequence of shortcut operations. Then the value of the minimum s^{out}-t edge-cut in G'' is at least OPT_s. Moreover, if (s,T) is a good pair, and all shortcut operations in the sequence are valid, then the value of the minimum s^{out}-t edge-cut in G'' is exactly $\mathsf{OPT}_s = \mathsf{OPT}$.*

Proof: From the Max-Flow/Min-Cut theorem, the value of the minimum s^{out}-t edge-cut in G' is equal to the value of the maximum s^{out}-t flow in G'. The insertion of edges into G' may not decrease the value of the maximum s^{out}-t flow. Therefore, the value of the minimum s^{out}-t edge-cut in G'' is at least OPT_s.

Assume now that (s,T) is a good pair, and that all shortcut opreations in the sequence are valid. Let $E_S = \left\{(v^{\text{in}}, v^{\text{out}}) \mid v \in S\right\}$ be the set of special edges in graph G' that represent the vertices of S. Let $X = L^* \cup S^{\text{in}}$ and let $Y = V(G') \setminus X$. Note that (X,Y) is an s^{out}-t cut in G'. Clearly, $E_{G'}(X,Y) = E_S$, and so the edges of E_S separate the vertices of X from the vertices of Y in G'. Moreover, since (L,S,R) is a minimum vertex-cut in G, $\sum_{e \in E_S} c(e) = \sum_{v \in S} w(v) = \mathsf{OPT}_s = \mathsf{OPT}$. Since $s^{\text{out}} \in X$ and $t \in Y$, and since the valid shortcut edges that we add all connect vertices of Y to t, set E_S of edges also separates X from Y in G''. Therefore, the value of the minimum s^{out}-t edge-cut in G'' is $\sum_{e \in E_S} c(e) = \mathsf{OPT}_s = \mathsf{OPT}$. □

For convenience, we denote by G'' the graph that our algorithm maintains by starting from G', and then gradually adding shortcut edges to it, while graph G' remains fixed throughout the algorithm. We use the first copy of the modified adjacency-list representation of G' that our algorithm receives as input in order to maintain G''. We mark in the modified adjacency-list representation of G'' every vertex v for which a shortcut edge (v,t) was added to G''. Our algorithm also maintains an s^{out}-t flow f in G'', by starting with $f = 0$, and then gradually augmenting it. Specifically, we maintain a set $E_f = \{e \in E(G'') \mid f(e) > 0\}$ of edges in an efficient search structure (such as a Binary Search Tree), and, for every edge $e \in E_f$, we list the current flow value $f(e)$. For convenience, we denote this data structure by DS_f. Lastly, we maintain a graph $H = G''_f$, the residual flow network of G'' with respect to f. Initially, when $G'' = G'$ and $f = 0$, graph H is identical to G'. We use the second copy of the modified adjacency-list representation of G' in order to maintain H. For every edge $e \in E(H)$, its capacity in H is denoted by $\hat{c}(e)$; we sometimes refer to $\hat{c}(e)$ as the *residual capacity* of e. Recall that, for an edge $e = (x,y) \in E(G'')$, if $f(e) < c(e)$, then a copy of e (that we refer to as its *forward copy*) is present in H, and $\hat{c}(e) = c(e) - f(e)$. Additionally, if $f(e) > 0$, then an anti-parallel edge (y,x) (that we refer to as the *backward copy of* e) is present in H, with $\hat{c}(y,x) = f(e)$. If edge $e \in E(G'')$ is special, then all its copies in H are special edges as well; otherwise, the copies of e in H are regular edges.

We emphasize that OPT_s is the value of the maximum s^{out}-t flow in the initial graph G'; the value of the s^{out}-t flow in G'' may be higher. Next, we define several parameters that are used throughout the algorithm.

Parameters. Throughout the algorithm, we use a parameter $\epsilon = \frac{1}{45}$. Recall that, if (s,T) is a good pair, then $|S| \geq n^{44/45} \cdot |L| = n^{1-\epsilon} \cdot |L|$ must hold, and so:

$$|L| \leq \frac{|S|}{n^{1-\epsilon}} \leq n^{\epsilon} \tag{3}$$

We assume that n is greater than a sufficiently large constant, so that, for example:

$$\epsilon > \frac{1}{\sqrt{\log n}} \tag{4}$$

holds. Additionally, we assume that:

$$n^{\epsilon/10} > 2^{20} \log^2 n \cdot \log^2(W_{\max}). \tag{5}$$

Notice that, if either of these assumptions does not hold, then we can use the algorithm from Theorem 2.4, in order to compute the value OPT_s of the maximum s^{out}-t flow in G', which is equal to the value of the minimum s-T_s vertex-cut in G, as observed above. The running time of the algorithm in this case is $O\left(n^{2+o(1)} \cdot \log(W_{\max})\right) \leq O\left(n \cdot (\log(W_{\max}))^{O(1)}\right)$, as required. Therefore, we assume from now on that n is large enough, and that, in particular, Inequalities 4 and 5 hold. Our second main parameter is γ, defined to be the smallest integral power of 2, so that $\gamma \geq n^{7/9}$ holds. It is immediate to verify that:

$$n^{7/9} \leq \gamma \leq 2n^{7/9} \tag{6}$$

and so:

$$n^{2/3+5\epsilon} \leq \gamma \leq 2n^{1-10\epsilon}, \tag{7}$$

and additionally:

$$\gamma \geq n^{14\epsilon}. \tag{8}$$

Our last main parameter is $\delta = 4\epsilon - \frac{\log(4000 \log n)}{\log n} = \frac{4}{45} - \frac{\log(4000 \log n)}{\log n} = \frac{4}{45} - o(1)$. This parameter is chosen so that:

$$n^{\delta} = \frac{n^{4\epsilon}}{4000 \log n} \tag{9}$$

holds, and it will be used as the parameter for the δ-subgraph oracle. Notice that from our assumptions:

$$\delta \geq \frac{1}{\sqrt{\log n}} \tag{10}$$

holds, as required by the algorithm for the subgraph oracle from Lemma 3.17.

Our algorithm consists of at most $z = \left\lceil \log\left(\frac{8W'_{\max}\cdot n^2}{\gamma}\right)\right\rceil \leq 4\log n\cdot\log(W'_{\max})$ phases. For all $0 \leq i \leq z$, we let $M_i = \frac{W'_{\max}}{2^i\cdot\gamma}$. Notice that $M_0 = \frac{W'_{\max}}{\gamma}$, and $M_z \leq \frac{1}{8n^2}$, and for all i, M_i is an integral power of 2.

Our algorithm may terminate at any time with a "FAIL"; in this case we output $c_s = W_{\max}$. For all $i \geq 1$, as long as the algorithm did not terminate with a "FAIL", at the beginning of the ith phase, it computes an s^{out}-t flow f_{i-1} in the current graph G'', such that the following invariants hold:

I1. With probability at least $1 - \frac{2i}{n^\epsilon\cdot\log^2(W_{\max})}$, all shortcut operation performed before the start of Phase i are valid;

I2. If (s,T) is a good pair, then the probability that the algorithm terminates prior to the beginning of Phase i with a "FAIL" is at most $\frac{10i^2-2i}{n^\epsilon\cdot\log^2(W_{\max})}$.

I3. Flow f_{i-1} is M_{i-1}-integral, and $\text{val}(f_{i-1}) \geq \mathsf{OPT}_s - 4nM_{i-1}$;

I4. The total number of edges $e \in E(G'')$ with $f_{i-1}(e) > 0$ is at most $O\left(i\cdot n^{2-4\epsilon+o(1)}\cdot\log^4(W_{\max})\right)$; and

I5. For every vertex $v \in V(G)$ with $w(v) \geq M_{i-1}$, the flow f_{i-1} on the corresponding special edge $(v^{\text{in}}, v^{\text{out}})$ is at most $w(v) - M_{i-1}$.

At the end of the last phase, the algorithm produces a flow f_z, for which Invariants I1–I5 hold as well. For consistency of notation, we will sometimes view the end of Phase z as the beginning of Phase $(z+1)$, even though the algorithm does not execute Phase $(z+1)$.

In addition to data structure DS_f for the current flow f, our algorithm also maintains the current graph G'' and the residual flow network H of G'' with respect to f, both in the modified adjacency-list representation. We note that shortcut operations may be performed over the course of each phase, and during the preprocessing step. In every phase, we make at most one call to the δ-subgraph oracle, and we make an additional call to the oracle in the preprocessing step. We ensure that the running time of every phase, and of the preprocessing step, is bounded by $O\left(n^{2-4\epsilon+o(1)}\cdot(\log(W_{\max}))^{O(1)}\right)$.

Bad events $\mathcal{E}_i$. For all $1 \leq i \leq z+1$, we let $\mathcal{E}_i$ be the bad event that either (i) at least one shortcut operation executed before the beginning of Phase i was invalid; or (ii) the algorithm terminated prior to the beginning of Phase i with a "FAIL". From the above invariants, if (s,T) is a good pair, then, for all $1 \leq i \leq z+1$:

$$\mathbf{Pr}\left[\mathcal{E}_i\right] \leq \frac{10i^2}{n^\epsilon\cdot\log^2(W_{\max})}. \tag{11}$$

We start by providing an algorithm for the preprocessing step, whose purpose is to compute the initial flow f_0, that will serve as input to the first phase, together with the corresponding residual flow network.

4.1 The Preprocessing Step

Our algorithm starts with $G'' = G'$. The purpose of the preprocessing step is to compute an initial s^{out}-t flow f_0 in G'' (after possibly performing some shortcut operations in G''), so that Invariants

I1–I5 hold at the beginning of Phase 1. We will also compute the residual flow network $H = G''_{f_0}$ in the modified adjacency list representation, and update the data structure DS_f to represent the flow $f = f_0$. Flow f_0 and graphs G'' and H will then serve as input to the first phase.

Let $\hat{U}_0 = \{v \in V(G) \mid w(v) \geq M_0\}$, and let $Z = \hat{U}_0 \setminus (\{s\} \cup N_G(s))$. Note that we can compute $\hat{U}_0$ and Z_0 in time $O(n)$ using the adjacency-list representation of G.

We start with intuition. Assume that (s, T) is a good pair, so $s \in L$ holds. Then, from Corollary 2.3, $|Z \cap S| \leq |L| \cdot \gamma$, and so at most $|L| \cdot \gamma + |L| \leq 2|L| \cdot \gamma$ vertices of Z may lie in $S \cup L$. Consider now any vertex $v \in \hat{U}_0$. We say that v is a *bad vertex*, if $|N_G(v) \cap Z| > 2|L| \cdot \gamma$, and otherwise we say that it is a *good vertex*. Notice that, if v is a bad vertex, then it must have a neighbor $u \in N_G(v)$ that lies in $Z \setminus (S \cup L) \subseteq R$, and therefore v may not lie in L. Adding the edge (v^{out}, t) to graph G'' is then a valid shortcut operation.

We will construct a sparsified subgraph $\hat{G} \subseteq G''$, as follows. We start by letting $\hat{G}$ be obtained from G'' by deleting from it all vertices in $\left\{v^{\mathrm{in}}, v^{\mathrm{out}} \mid v \in V(G) \setminus \left(\hat{U}_0 \cup \{s\}\right)\right\}$. We then decrease the capacity of every special edge e in the resulting graph by at least M_0 and at most $2M_0$ units, so that the resulting new capacity $c'(e)$ satisfies $\max\{c(e) - 2M_0, 0\} \leq c'(e) \leq c(e) - M_0$, and it is an integral multiple of M_0 (which may be equal to 0). Next, for every vertex $v \in \hat{U}_0$, if v is a bad vertex, then we add the shortcut edge (v^{out}, t) to $\hat{G}$ (and to G''), and we delete all other edges leaving v^{out} from $\hat{G}$. If v is a good vertex, then we delete all edges connecting v^{out} to the in-copies of the vertices in $N_G(s)$. Consider now the resulting graph $\hat{G}$. On the one hand, this graph is quite sparse: it only contains copies of vertices $v \in \hat{U}_0 \cup \{s\}$ (in addition to the vertex t), and, for each such vertex v, its in-copy has 1 outgoing edge, and its out-copy has at most $2|L| \cdot \gamma$ outgoing edges in $\hat{G}$, so $|E(\hat{G})| \leq O(n \cdot |L| \cdot \gamma)$ holds. On the other hand, it is not hard to see (and we prove it formally below) that there is an s^{out}-t flow in $\hat{G}$ of value at least $\mathsf{OPT}_s - 4n \cdot M_0$. Since M_0 and $W'_{\max}$ are both powers of 2, we can ensure that this flow is M_0-integral, and it must satisfy Invariant I5, since we decreased the capacity of every special edge in $\hat{G}$ by at least M_0 flow units. By computing the maximum s^{out}-t flow in graph $\hat{G}$ we then obtain the desired flow f_0.

Our algorithm follows the above high-level plan, except for minor changes that are required in order to ensure that it is efficient. There are two main difficulties with the plan we have described: (i) computing the set of all bad vertices efficiently; and (ii) constructing the graph $\hat{G}$ efficiently. We overcome both these challenges by employing the subgraph oracle.

We now describe the preprocessing step formally. Recall that we denoted by $\hat{U}_0 \subseteq V(G)$ the set of all vertices $v \in V(G)$ with $w(v) \geq M_0 = \frac{W'_{\max}}{\gamma}$, and we denoted by $Z = \hat{U}_0 \setminus (\{s\} \cup N_G(s))$. We start by querying the δ-subgraph oracle with the graph G and the set Z of its vertices. Let (Y^h, Y^ℓ, E') denote the response of the oracle. Recall that $|E'| \leq n^{2-\delta} \cdot \log^2(W_{\max}) \leq O\left(n^{2-4\epsilon} \cdot \log n \cdot \log^2(W_{\max})\right)$ holds (since $n^\delta = \frac{n^{4\epsilon}}{4000 \log n}$ from Equation 9), and that $E' = E_G(Y^\ell, Z)$. Let $\mathcal{E}'_0$ be the bad event that the oracle erred. From the definition of the δ-subgraph oracle, the probability that it errs is at most $\frac{1}{n^5 \cdot \log^4(W_{\max})}$, so $\mathbf{Pr}\left[\mathcal{E}'_0\right] \leq \frac{1}{n^5 \cdot \log^4(W_{\max})}$. We use the following observation.

Observation 4.2 *If Event $\mathcal{E}'_0$ did not happen, and if (s, T) is a good pair, then $Y^h \subseteq S \cup R$ must hold.*

Proof: Assume that Event $\mathcal{E}'_0$ did not happen, and that (s, T) is a good pair. Consider some vertex $v \in Y^h$. Since we have assumed that the oracle did not err:

$$\begin{aligned}|N_G(v) \cap Z| &\geq \frac{n^{1-\delta}}{1000 \log n} \\ &= 4n^{1-4\epsilon} \\ &> 2n^{\epsilon} \cdot \gamma \\ &\geq 2|L| \cdot \gamma\end{aligned}$$

since $n^{\delta} = \frac{n^{4\epsilon}}{4000 \log n}$ from Equation 9, $\gamma < 2n^{1-5\epsilon}$ from Inequality 7, and $|L| \leq n^{\epsilon}$ from Inequality 3.

Observe that, from Corollary 2.3, since all vertex weights in G are bounded by $W'_{\max}$ and $M_0 = \frac{W'_{\max}}{\gamma}$, $|Z \cap S| \leq |L| \cdot \gamma$, and so at most $|L| \cdot \gamma + |L| < 2|L| \cdot \gamma$ vertices of Z may lie in $S \cup L$. Since v has at least $2|L| \cdot \gamma$ neighbors in Z, there is some vertex $u \in Z \setminus (S \cup L)$, that is a neighbor of v in G. But then $u \in R$ must hold, and so $v \in S \cup R$. We conclude that $Y^h \subseteq S \cup R$. □

We perform the following sequence of shortcut operations: for every vertex $v \in Y^h$, we add the edge (v^{out}, t) to G'', and we modify the adjacency-list representation of G'' accordingly. From Observation 4.2, if bad event $\mathcal{E}'_0$ did not happen, then all shortcut operations that we perform are valid. From now on the algorithm for the preprocessing step may not terminate with a "FAIL", and it will not perform any additional shortcut operations. Since $\mathbf{Pr}\left[\mathcal{E}'_0\right] \leq \frac{1}{n^5 \cdot \log^4(W_{\max})}$, this ensures that Invariants I1 and I2 will hold at the beginning of Phase 1.

Next, we construct a subgraph $\hat{G} \subseteq G''$. We will show that $|E(\hat{G})|$ is sufficiently small, and we will provide an algorithm that constructs $\hat{G}$ efficiently. We will also show that the value of the maximum s^{out}-t flow in $\hat{G}$ is at least $\mathsf{OPT}_s - 4n \cdot M'$. Lastly, we will compute a maximum s^{out}-t flow in $\hat{G}$ that is M_0-integral, obtaining the desired flow f_0, and the corresponding residual flow network H. We start by defining the graph $\hat{G}$.

Definition of $\hat{G}$. We start by setting $\hat{G} = G''$. We then delete from $\hat{G}$ the copies of all vertices in $V(G) \setminus \left(\hat{U}_0 \cup \{s\}\right)$, and all edges that are incident to them. We reduce the capacity of each remaining special edge e by at least M_0 and at most $2M_0$, so that the resulting new capacity $c'(e)$ satisfies $\max\{c(e) - 2M_0, 0\} \leq c'(e) \leq c(e) - M_0$, and it is an integral multiple of M_0 (which may be equal to 0). We denote by $E_1^{\text{del}} \subseteq E(G'')$ all edges that were deleted from $\hat{G}$ in this step. If v is a vertex of $\hat{U}_0 \cup \{s\}$, then we imagine splitting the special edge $e_v = (v^{\text{in}}, v^{\text{out}})$ of G'' into two copies. The first copy, denoted by e'_v, has capacity $c(e_v) - c'(e_v)$ and is added to E_1^{del}; the second copy, denoted by e''_v, has capacity $c'(e_v)$, and is viewed as the special edge representing v in graph $\hat{G}$.

Next, for every vertex $v \in Y^h \setminus \{s\}$, we delete all edges leaving v^{out} from the resulting graph, except for the edge (v^{out}, t). We denote the set of all edges deleted in this step by E_2^{del}. Lastly, for every vertex $v \in \hat{U}_0 \setminus \{s\}$, we delete all edges connecting v^{out} to the vertices in the set $\left\{u^{\text{in}} \mid u \in N_G(s) \cup \{s\}\right\}$, and we denote by E_3^{del} the set of all edges deleted in this step. This completes the definition of the graph $\hat{G}$. Let $E^{\text{del}} = E_1^{\text{del}} \cup E_2^{\text{del}} \cup E_3^{\text{del}}$. Then $E(\hat{G}) = E(G'') \setminus E^{\text{del}}$.

Efficient Construction of $\hat{G}$. We now provide an efficient algorithm for constructing the graph $\hat{G}$. First, for every vertex $v \in \hat{U}_0 \cup \{s\}$, we include vertices $v^{\text{in}}, v^{\text{out}}$, and the special edge $e''_v = (v^{\text{in}}, v^{\text{out}})$ of capacity $c'(e_v)$ as defined above. We also add vertex t, and all regular edges that are incident to t or to s^{out} in G'', whose other endpoint was already added to $\hat{G}$. Lastly, for every edge $e = (u, v) \in E'$ (here, E' is the set of edges returned by the subgraph oracle), if $u \notin Y^h$, then we add the regular edge $(u^{\text{out}}, v^{\text{in}})$ to $\hat{G}$, and, if $v \notin Y^h$, then we add the regular edge $(v^{\text{out}}, u^{\text{in}})$ to $\hat{G}$. The capacities

of all these edges are the same as in G''. It is easy to verify that the graph that we obtain is indeed $\hat{G}$, that $|E(\hat{G})| \leq O(n) + O(|E'|) \leq O\left(n^{2-4\epsilon} \cdot \log n \cdot \log^2(W_{\max})\right)$, and that $\hat{G}$ can be computed in time $O(n) + O(|E'|) \leq O\left(n^{2-4\epsilon} \cdot \log n \cdot \log^2(W_{\max})\right)$. Notice that the capacity of every edge in $\hat{G}$ is an integral multiple of M_0.

Maximum Flow in $\hat{G}$. We compute the maximum s^{out}-t flow in $\hat{G}$, using the algorithm from Theorem 2.4, where the resulting flow, denoted by f_0, is given via the edge-representation, and it is M_0-integral (since all edge capacities in $\hat{G}$ are M_0-integral). The running time of the algorithm is:

$$O\left(|E(\hat{G})|^{1+o(1)} \cdot \log^2(W_{\max})\right) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^5(W_{\max})\right).$$

We then update the modified adjacency-list representation of H, so that it becomes a valid modified adjecency-list representation of the residual flow network of G'' with respect to f_0, and the data structure DS_f to represent the flow f_0. It is easy to verfy that both can be done in time $O\left(n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right)$. This completes the description of the algorithm for the preprocessing step. We now complete its analysis.

Running time analysis. The sets $\hat{U}_0$ and Z of vertices of G can be computed in time $O(n)$ using the adjacency list representation of graph G. The time that is required to compute the graph $\hat{G}$, to compute the flow f_0, and to update the residual flow network H is bounded by $O\left(n^{2-4\epsilon+o(1)} \cdot \log^5(W_{\max})\right)$. Therefore, the total running time of the preprocessing step is:

$$O\left(n^{2-4\epsilon+o(1)} \cdot \log^5(W_{\max})\right).$$

We have already established that, at the beginning of Phase 1, Invariants I1 and I2 hold. Invariant I5 follows immediately from the definition of the flow network $\hat{G}$, since for every vertex $v \in M_0$, the capacity of its corresponding edge $(v^{\text{in}}, v^{\text{out}})$ in $\hat{G}$ is set to be at most $w(v) - M_0$. Notice that the total number of edges $e \in E(G'')$ with $f_0(e) > 0$ is bounded by $|E(\hat{G})| \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right)$, so Invariant I4 holds as well. We use the following claim to prove that Invariant I3 also holds.

Claim 4.3 $\text{val}(f_0) \geq \mathsf{OPT}_s - 4n \cdot M_0$.

Proof: Consider the graph G'' that is obtained at the end of the preprocessing step. For convenience, we take again the view that, for every vertex $v \in \hat{U}_0$, its corresponding special edge $e_v = (v^{\text{in}}, v^{\text{out}})$ in G'' is replaced by two parallel copies: copy e'_v of capacity $c(e_v) - c'(e_v) \leq 2M_0$, and copy e''_v of capacity $c'(e_v)$. This splitting of special edges does not change the value of the maximum s^{out}-t flow in G''.

From Claim 4.1, there is an s^{out}-t flow f in G'' of value at least OPT_s. Consider a flow-path decomposition of f, and let $\mathcal{P}$ be the resulting collection of paths that carry non-zero flow. We can assume w.l.o.g. that every path $P \in \mathcal{P}$ contains at most one vertex in set $B = \left\{v^{\text{in}} \mid v \in N_G(s)\right\}$, and that this vertex is the second vertex on the path. Indeed, if this is not the case, then we can "shortcut" the path P by connecting s^{out} directly to the last vertex of B on the path P. Therefore, in particular, the paths in $\mathcal{P}$ may not contain edges of E_3^{del}.

We partition $\mathcal{P}$ into two subsets: set $\mathcal{P}_1$ contains all paths P with $P \subseteq \hat{G}$, and $\mathcal{P}_2$ contains all remaining paths.

Consider now some path $P \in \mathcal{P}_2$. Since $P \not\subseteq \hat{G}$, at least one edge of P must lie in E^{del}. Let $e(P)$ be the first such edge on P. As discussed, already, $e(P) \notin E_3^{\text{del}}$, so either $e(P) \in E_1^{\text{del}}$, or $e(P) \in E_2^{\text{del}}$

must hold. We further partition $\mathcal{P}_2$ into two subsets $\mathcal{P}_2^1$ and $\mathcal{P}_2^2$, where a path $P \in \mathcal{P}_2$ belongs to the former set if $e(P) \in E_1^{\text{del}}$, and to the latter set otherwise.

Consider first a path $P \in \mathcal{P}_2^2$, and recall that the corresponding edge $e(P)$ must originate at a vertex v^{out} for some $v \in Y^h$. Moreover, our algorithm inserted the edge (v^{out}, t) of capacity $W_{\max}$ into G'', and this edge lies in $\hat{G}$ as well. We replace the path P with a new path $P' \subseteq \hat{G}$, that follows P from s^{out} to v^{out}, and then uses the edge (v^{out}, t). We send $f(P') = f(P)$ flow on path P'. By using the flow-paths in $\mathcal{P}_1$ and in $\left\{P' \mid P \in \mathcal{P}_2^2\right\}$, we obtain a valid s^{out}-t flow in $\hat{G}$, whose value is $\text{val}(f) - \sum_{P \in \mathcal{P}_2^1} f(P)$.

It is easy to verify that $|E_1^{\text{del}}| \leq n$, since the set E_1^{del} contains one special edge for each vertex $v \in V(G)$. Moreover, the capacity of every edges in E_1^{del} is bounded by $2M_0$. Therefore, $\sum_{P \in \mathcal{P}_2^1} f(P) \leq 2n \cdot M_0$, and so, from the above discussion, the value of the maximum s^{out}-t flow in $\hat{G}$ is at least $\text{val}(f) - \sum_{P \in \mathcal{P}_2^1} f(P) \geq \text{val}(f) - 2n \cdot M_0$. □

4.2 Finishing the Algorithm

So far we have provided an algorithm for the preprocessing step, that computes the initial flow f_0, and the corresponding residual flow network H, for which Invariants I1–I5 hold. The running time of the preprocessing step is $O\left(n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right)$.

The algorithm for implementing each phase is the most invovled technical part of our paper, and we provide it below. We assume for now that there exists an algortihm for implementing each phase, with the following properties:

- At the end of each phase, Invariants I1–I5 hold;
- The algorithm performs at most one call to the δ-subgraph oracle in every phase; and
- The running time of the algorithm for a single phase is bounded by $O\left(n^{2-4\epsilon+o(1)} \cdot (\log(W_{\max}))^{O(1)}\right)$.

We provide an algorithm for executing each phase below, after we complete the proof of Theorem 3.16 under these assumptions.

Recall that the number of phases in the algorithm is $z = \left\lceil \log\left(\frac{8W'_{\max} \cdot n^2}{\gamma}\right)\right\rceil \leq 4 \log n \cdot \log(W'_{\max})$, and that $M_z \leq \frac{1}{8n^2}$. The total running time of all z phases and the preprocessing step is then bounded by:

$$O\left(z \cdot n^{2-4\epsilon+o(1)} \cdot (\log(W_{\max}))^{O(1)}\right) \leq O\left(n^{86/45+o(1)} \cdot (\log(W_{\max}))^{O(1)}\right),$$

since $\epsilon = 1/45$. The number of times the algorithm accesses the δ-subgraph oracle is bounded by $z + 1 \leq 8 \log n \cdot \log(W_{\max})$.

If our algorithm ever terminates with a "FAIL", then we return $c_s = W_{\max}$. Clearly, c_s is at least as high as the value of the minimum s-T_s vertex cut in G. Let $\mathcal{E}'$ be the bad event that (s, T) is a good pair, and the algorithm either terminated with a "FAIL", or at least one of the shortcut operations that it performed was invalid. From Invariants I1 and I2:

$$\mathbf{Pr}\left[\mathcal{E}'\right] \leq \frac{10(z+1)^2}{n^{\epsilon} \cdot \log^2(W_{\max})} \leq \frac{640 \log^2 n}{n^{\epsilon}} \leq \frac{1}{16},$$

from Inequality 5. Notice that, from Invariant I3, $\mathrm{val}(f_z) \geq \mathsf{OPT}_s - 4n \cdot M_z \geq \mathsf{OPT}_s - \frac{1}{2n}$. Let $E' = E_{f_z}$ be the collection of all edges $e \in E(G'')$ with $f_z(e) > 0$. From Invariant I4, $|E'| \leq O\left(z \cdot n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^5(W_{\max})\right)$.

We let $G^* \subseteq G''$ be the subgraph of G'' that only contains the edges of E'. The last step in our algorithm is to compute a maximum s^{out}-t flow in G^*, using the algorithm from Theorem 2.4. We then output $c_s = \mathrm{val}(f^*)$. Observe that the running time of this last step is:

$$O(|E'|^{1+o(1)} \cdot \log(W_{\max})) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^7(W_{\max})\right) \leq O\left(n^{86/45+o(1)} \cdot \log^7(W_{\max})\right),$$

since $\epsilon = 1/45$, and so the total running time of the algorithm is $O\left(n^{86/45+o(1)} \cdot (\log(W_{\max}))^{O(1)}\right)$.

Since all edge capacities in G^* are integral, flow f^*, and its value $\mathrm{val}(f^*)$ are integral. Since OPT_s is an integer, and since $\mathrm{val}(f_z) \geq \mathsf{OPT}_s - \frac{1}{2n}$, we get that $c_s = \mathrm{val}(f^*) \geq \lceil \mathrm{val}(f_z) \rceil \geq \mathsf{OPT}_s$.

Lastly, if (s, T) is a good pair, and if Event $\mathcal{E}'$ did not happen, then all shortcut operations that the algorithm performed were valid, and so, from Claim 4.1, the value of the maximum s^{out}-t flow in G'', and hence in G^*, is bounded by OPT_s. Since $\mathbf{Pr}\left[\mathcal{E}'\right] < \frac{1}{2}$, we get that, if (s, T) is a good pair, then, with probability at least $\frac{1}{2}$, c_s is equal to the value of the minimum s-T_s vertex-cut in G.

It now remains to provide an algorithm for executing each phase, which we do next.

4.3 The Algorithm for the ith Phase

We now provide the description of the ith phase, for $1 \leq i \leq z$. For convenience, we denote $M' = M_i = \frac{W'_{\max}}{2^i \cdot \gamma}$, and we denote by $\hat{U}_i \subseteq V(G)$ the set of all vertices $v \in V(G)$ with $w(v) \geq M'$. We assume that, at the beginning of the phase, Invariants I1–I5 hold. In particular, we are given an s^{out}-t flow f_{i-1} in the current graph G'', that is $M_{i-1} = (2M')$-integral, and $\mathrm{val}(f_{i-1}) \geq \mathsf{OPT}_s - 4n \cdot M_{i-1} = \mathsf{OPT}_s - 8n \cdot M'$ holds. Additionally, we are given the residual flow network H for the current graph G'', with respect to the flow f_{i-1}. From Invariant I5, the residual flow network H has the following property:

- Q1. For every vertex $v \in V(G)$ with $w(v) \geq M_{i-1} = 2M'$, there is a special edge $e = (v^{\mathrm{in}}, v^{\mathrm{out}})$ in H, whose residual capacity $\hat{c}(e) \geq M_{i-1} = 2M'$.

Notice that, in particular, from the above property, and since the flow f_{i-1} is $2M'$-integral, for every vertex $v \in \hat{U}_i$, the residual capacity of the corresponding special edge $(v^{\mathrm{in}}, v^{\mathrm{out}})$ in H is at least M'. We further partition $\hat{U}_i$ into two subsets: set $\hat{U}_i^h$ containing all vertices v for wich the corresponding special edge $e = (v^{\mathrm{in}}, v^{\mathrm{out}})$ has residual capacity $\hat{c}(e) \geq M' \cdot \gamma$ in H, and set $\hat{U}_i^\ell = \hat{U}_i \setminus \hat{U}_i^h$.

Let $\Delta = \mathsf{OPT}_s - \mathrm{val}(f_{i-1})$, so $\Delta \leq 8n \cdot M'$. By the properties of the residual flow network, there is an s^{out}-t flow in H of value Δ. Moreover, if (s, T) is a good pair, and if all shortcut operations so far were valid, then the value of the maximum s^{out}-t flow in H is exactly Δ.

We now provide the intuition for the algorithm for the ith phase. For the sake of the intuition, assume that (s, T) is a good pair. The goal of the ith phase is to compute an s^{out}–t flow f_i in H of value at least $\Delta - 4n \cdot M'$, after possibly performing some shortcut operations, such that the flow is M'-integral. In order to do so, like in the preprocessing step, we first sparsify the graph H, and then compute the flow in the sparsified graph. A major step towards the sparsification of H is to compute a partition (Q, Q') of $\hat{U}_i$ into two subsets, such that Q' only contains relatively few vertices of $S \cup L$. Additionally, if $|Q| > n^{1-4\epsilon}$, then we will also compute an s^{out}-t cut $(\hat{X}, \hat{Y})$ in H, such that, in any maximum s^{out}-t flow in H, the total flow over the edges in $E_H(\hat{X}, \hat{Y})$ is close to their capacity, and for every vertex

$v \in Q$, $v^{\text{in}} \in \hat{X}$ and $v^{\text{out}} \in \hat{Y}$ holds. Intuitively, if a vertex $v \in \hat{U}_i$ has many edges connecting it to the vertices of Q', then we are guaranteed that $v \in S \cup R$, and therefore, we can add a shortcut edge (v^{out}, t), and delete all other edges leaving v^{out} in the sparsified graph. If a vertex $v \in \hat{U}_i$ only has few edges connecting it to vertices of Q', then we can include all the corresponding edges of H in the sparsified subgraph, but we will need to further sparsify the edges connecting copies of such vertices to the vertices of $Q^* = Q^{\text{in}} \cup Q^{\text{out}}$, if $|Q| > n^{1-4\epsilon}$. For this last sparsification step, we exploit the cut $(\hat{X}, \hat{Y})$ that our algorithm computes in this case. In order to compute the partition (Q, Q') of $\hat{U}_i$ with the desired properties, we start by constructing an auxiliary graph $J \subseteq H$ with $s^{\text{out}} \in V(J)$, $t \notin V(J)$, and which has the additional property that for the vast majority of vertices $v \in S$, $v^{\text{in}} \in V(J)$ and $v^{\text{out}} \notin V(J)$ holds. We then let $\tilde{H}$ be a graph that is obtained from H by contracting all vertices of $V(H) \setminus V(J)$ into a new sink vertex t'. We show that the desired partition (Q, Q') of $\hat{U}_i$ can be computed by performing a series of min-cost flow computation in $\tilde{H}$. However, in order to ensure that these min-cost flow computations can be executed efficiently, we need to ensure that the graph $\tilde{H}$ itself is sufficiently sparse. Our algorithm for Phase i then consists of three steps:

Step 1. Perform iterative local flow computations, whose purpose is to ensure that the graph $\tilde{H}$ constructed at the end of this step is sufficiently sparse. During this step we may perform some shortcut operations, and augment the current flow f_{i-1}, obtaining a new s^{out}-t flow f in G''. This step is described in Section 5.

Step 2. Compute a partition (Q, Q') of $\hat{U}_i$ by executing a number of min-cost flow computations in $\tilde{H}$, and compute the cut $(\hat{X}, \hat{Y})$ in $\tilde{H}$ if $|Q| > n^{1-4\epsilon}$. This step is described in Section 6.

Step 3. Sparsify the graph H and compute the maximum s^{out}-t flow in the resulting flow network. We also augment the current flow f, to obtain the final flow f_i. This step is described in Section 7.

Notation. For brevity of notation, we denote $\hat{U}_i$ by U, and the sets $\hat{U}_i^h$ and $\hat{U}_i^\ell$ of vertices by U^h and U^ℓ, respectively. We start with the flow $f = f_{i-1}$, and, as the algorithm in this phase progresses, we may augment the flow f, which may lead to changes in the residual flow network. The sets U^h and U^ℓ of vertices are always defined with respect to the current flow network H. Recall that we denoted by $V^{\text{in}} = \left\{v^{\text{in}} \mid v \in V(G)\right\}$ and by $V^{\text{out}} = \left\{v^{\text{out}} \mid v \in V(G)\right\}$.

We now describe each of the steps in turn.

5 Step 1: Local Flow Augmentations

Our algorithm for this step relies on the technique of local flow augmentation, that was first introduced by [CHI+17] in the context of the Maximal 2-Connected Subgraph problem. The technique was later refined and applied to *unweighted* and *approximation* versions of the global minimum vertex-cut problem [NSY19, FNY+20, CQ21]. The algorithm in this section further expands and generalizes this technique, in particular by allowing local flow augmentations to be performed via general flows rather than just paths.

In order to provide intuition for this step, we consider a subgraph $H' \subseteq H$, that is defined via the following algorithmic process. Initially, graph H' contains only the vertex s^{out}, and then we iteratively expand it using the following two rules:

R1. If there exists an edge $e = (u, v) \in E(H)$ whose capacity $\hat{c}(e) \geq M' \cdot \gamma$, such that $u \in V(H')$ and $v \notin V(H')$, then add the edge e and the vertex v to H'.

R2. If there exists an integer $1 \leq j \leq \log \gamma$, a vertex $v^{\text{out}} \in V^{\text{out}} \setminus V(H')$, and a collection $Y \subseteq V(H') \cap V^{\text{in}}$ of exactly 2^j distinct vertices, such that, for every vertex $u^{\text{in}} \in Y$, a regular edge $(u^{\text{in}}, v^{\text{out}})$ of capacity at least $\frac{M' \cdot \gamma}{2^j}$ is present in H, add v^{out} to H', and, for all $u^{\text{in}} \in Y$, add the edge $(u^{\text{in}}, v^{\text{out}})$ to H'.

As we will show later, if graph H' is constructed via the above rules, then for every vertex $v \in V(H')$, there is an s^{out}-v flow in H' of value at least $\frac{M' \cdot \gamma}{2}$. Let $\Gamma = V(H') \cap V^{\text{out}}$. Intuitively, the goal of the current step is to ensure that $|\Gamma|$ is small. In order to achieve this, we will gradually augment the flow f, that is initially set to $f = f_{i-1}$, which, in turn, will lead to changes in the graphs H and H'. Once we ensure that $|\Gamma|$ is sufficiently small, we will construct a new graph $\tilde{H} \subseteq H$, obtained from H by unifying all vertices of $V(H) \setminus V(H')$ into a destination vertex t', which, in turn, will allow us, in Step 2, to compute the partition (Q, Q') of U with the desired properties. We will ensure that the resulting graph $\tilde{H}$ is sufficiently small.

We use a parameter $N = \frac{32n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}$. As long as $|\Gamma| \geq N$, our algorithm chooses a random vertex $v^{\text{out}} \in \Gamma$, computes a flow f' of value at least $\frac{M' \cdot \gamma}{2}$ in H' from s^{out} to v^{out}, performs a shortcut operation by adding the edge (v^{out}, t) to graphs G'' and H, and then augments the current flow f with the flow f' (that is extended via the edge (v^{out}, t), so it terminates at t). The residual flow network H is then updated with the new flow, and graph H' is recomputed. As long as $|\Gamma|$ is sufficiently large, with high probability, the resulting shortcut operation is valid. Since, in each such iteration, we augment the flow by at least $\frac{M' \cdot \gamma}{2}$ flow units, while we are guaranteed that, from Invariant I3, $\mathsf{OPT}_s - \text{val}(f_{i-1}) \leq 4nM_{i-1} = 8nM'$, the number of such iterations is bounded by $z' = \left\lceil \frac{32n}{\gamma} \right\rceil$. In order to ensure that the running time of every iteration is sufficiently low, we do not construct the entire graph H'; we terminate its construction once $|V(H') \cap V^{\text{out}}|$ reaches N. Consider any vertex $v^{\text{out}} \in V^{\text{out}}$ that does not currently lie in H', and an integer $1 \leq j \leq \gamma$. For brevity, we say that vertex v^{out} is *j-bad*, if there is a subset $Y \subseteq V^{\text{in}} \cap V(H')$ of 2^j distinct vertices, such that, for every vertex $u^{\text{in}} \in Y$, a regular edge $(u^{\text{in}}, v^{\text{out}})$ of capacity at least $\frac{M' \cdot \gamma}{2^j}$ lies in H. The execution of Rule R2 requires an algorithm that can correctly identify a j-bad vertex, if such a vertex exists for any $1 \leq j \leq \log \gamma$. While our algorithm can do this efficiently for relatively small values of the parameter j, for higher such values, for the sake of efficiency, we will employ the algorithm for the Heavy Vertex Problem from Claim 2.15. This, however, may lead to some imprecisions: for example, the algorithm may terminate the construction of graph H' while some j-bad vertex v still exists, for some sufficiently large value of j. However, in this case, we are guaranteed that, if $Y \subseteq V^{\text{in}} \cap V(H')$ is a set of vertices, such that, for every vertex $u^{\text{in}} \in Y$, a regular edge $(u^{\text{in}}, v^{\text{out}})$ of capacity at least $\frac{M' \cdot \gamma}{2^j}$ lies in H, then $|Y| \leq O(2^j \cdot \log n \cdot \log(W_{\max}))$ holds. Therefore, the graph that our algorithm constructs in the last iteration may be a subgraph of the graph H' as defined above, but its properties are sufficient for our purposes.

We now turn to provide a formal description of the algorithm for Step 1. The algorithm starts with the flow $f = f_{i-1}$, and then gradually augments it over the course of at most $z' = \left\lceil \frac{32n}{\gamma} \right\rceil$ iterations, where in every iteration the flow is augmented by $\frac{M' \cdot \gamma}{2}$ flow units. Throughout, we will ensure that the flow f remains M'-integral, and additionally, the following property holds:

Q'1. For every vertex $v \in V(G)$ with $w(v) \geq M'$, there is a special edge $(v^{\text{in}}, v^{\text{out}})$ in H, whose residual capacity is at least M'.

At the beginning of the algorithm, we set $f = f_{i-1}$. As observed already, this flow is $(2M')$-integral, and hence M'-integral. Moreover, since the flow is $(2M')$-integral, and since Property Q1 holds for it,

Property Q'1 must hold as well. Indeed, consider a vertex $v \in V(G)$ with $w(v) \geq M'$. If $w(v) \geq 2M'$, then, from Property Q1, the residual capacity of the special edge $(v^{\text{in}}, v^{\text{out}})$ in H is at least $2M' \geq M'$. Otherwise, since the flow f_{i-1} is $(2M')$-integral, no flow may be sent along the special edge $(v^{\text{in}}, v^{\text{out}})$, so its residual capacity remains $w(v) \geq M'$. We now provide an algorithm for executing a single iteration.

5.1 The Execution of a Single Iteration

As input to the qth iteration, we receive access to the modified adjacency list representation of the current graph G'', the current s^{out}-t flow f in G'' (that was obtained by augmenting the flow f_i), together with access to the modified adjacency-list representation of the residual flow network $H = G''_f$. We assume that the flow f is M'-integral, and that Property Q'1 holds for it. Recall that we have defined a parameter $N = \frac{32n^{1+2\epsilon} \cdot \log^2(W\max)}{\gamma}$. Let $\tau^* = \frac{n}{\sqrt{\gamma}}$, and let $j^* = \lceil \log(\tau^*) \rceil$. Throughout, we denote by $\hat{T} \subseteq V(H)$ the set of vertices $v \in V(H)$ for which the edge (v, t) is present in H.

We are now ready to describe an algorithm for a single iteration of Step 1. The main subroutine of the algorithm is Procedure Explore H', that is formally described in Figure 3. The procedure gradually constructs the graph H' using Rules R1 and R2, denoting by X^* the set of vertices of H' that were already discovered by the algorithm. It terminates once either (i) $|X^* \cap V^{\text{out}}| \geq N$ holds; or (ii) a vertex of $\hat{T}$ was reached; or (iii) the construction of the graph is complete. As mentioned already, in the latter case, the resulting graph may be a subgraph of the graph H' defined via the rules R1 and R2, but some j-bad vertices may still exist. We denote by $J \subseteq H'$ the subgraph of H' that the procedure constructs. In addition to the set $X^* \subseteq V(H')$ of vertices that were already discovered, the algorithm maintains a set $X \subseteq X^*$ of vertices that it still needs to explore, and the counter $N' = |X^* \cap V^{\text{out}}|$. Finally, for every vertex $v^{\text{out}} \in V^{\text{out}} \setminus V(J)$, for all $1 \leq j \leq j^*$, the algorithm maintains a counter $n_j(v^{\text{out}})$, that counts the number of regular edges $(u^{\text{in}}, v^{\text{out}})$ with $u^{\text{in}} \in X^*$ whose capacity is at least $\frac{M' \cdot \gamma}{2^j}$, that the algorithm discovered. Once this counter reaches value 2^j, vertex v^{out} is added to X and to X^*, and the counter is no longer maintained. All such edges $(u^{\text{in}}, v^{\text{out}})$ that the algorithm discovers prior to adding v^{out} to X are stored together with v^{out}, and, once v^{out} is added to X, all these edges are added to graph J. The formal description of Procedure Explore H' appears in Figure 3.

Let J' be the graph that is identical to J, except that, for every special edge e that lies in J, we decrease its capacity $\hat{c}(e)$ in J' by at least M' units and at most $2M'$ units, so that the new capacity $\hat{c}'(e)$ becomes an integral multiple of M', and $\hat{c}(e) - 2M' \leq \hat{c}'(e) \leq \hat{c}(e) - M'$ holds. (Note that the capacity of each special edge in J must be at least $\gamma \cdot M'$, from the description of Procedure Explore H'). Observe that now all edge capacities in J' are integral multiples of M'. Indeed, the above transformation ensures that the capacity of every special edge in J' is an integral multiple of M', and M' is an integral power of 2 from the definition. Since the capacity of every regular edge in G'' is $W_{\max}$, an integral power of 2, we get that it is an integral multiple of M'. Lastly, since the flow f is M'-integral, we get that the capacity of every regular edge in J' is an integral multiple of M'. We use the following claim.

Claim 5.1 *For every vertex v that is ever added to graph J by Procedure* Explore H'*, there is an s^{out}-v flow in J' of value at least $\frac{M' \cdot \gamma}{2}$, that is M'-integral. Moreover, after f is augmented with this flow, Property Q'1 continues to hold.*

Proof: It is sufficient to prove that, for every vertex v that lies in J, there is an s^{out}-v flow in J' of value at least $\frac{M' \cdot \gamma}{2}$. Indeed, if this is the case then, since all edge capacities in J' are integral multiples of M', we get that there is an s^{out}-v flow in J' of value at least $\frac{M' \cdot \gamma}{2}$ that is M'-integral.

PROCEDURE Explore H'

1. Initialize the data structures:
   (a) Set $X^* = X = \{s^{\text{out}}\}$, $N' = 0$.
   (b) Initialize the graph J to contain the vertex s^{out} only.
   (c) For all $1 \leq j \leq j^*$, for every vertex $v^{\text{out}} \in V^{\text{out}}$, set $n_j(v^{\text{out}}) = 0$.
2. While $X \neq \emptyset$, do:
   (a) Let $x \in X$ be any vertex. Delete x from X.
   (b) If $x \in \hat{T}$: terminate the algorithm and return x.
   Assume from now on that $x \notin \hat{T}$.
   (c) For every edge $e = (x, y) \in E(H)$ with $y \notin X^*$, do:
      i. If the capacity of e in H is at least $M' \cdot \gamma$:
         - add y to X and to X^*;
         - add y and the edge (x, y) to J;
         - if $y \in V^{\text{out}}$, then increase N' by 1, and, if $N' = N$ holds, halt.
      ii. Otherwise, if $x \in V^{\text{in}}$, $y \in V^{\text{out}} \setminus X^*$, and e is a regular edge, then, for all $1 \leq j \leq j^*$, such that the capacity of e in H is at least $\frac{M' \cdot \gamma}{2^j}$ do:
         - increase $n_j(y)$ by 1 and store the edge (x, y) with y.
         - if $n_j(y) = 2^j$, then add y to X and to X^*; add y and the 2^j regular edges of capacity at least $\frac{M' \cdot \gamma}{2^j}$ stored with it to J; increase N' by 1, and, if $N' = N$ holds, halt.
3. For all $j^* < j \leq \log \gamma$, do:
   (a) Execute the algorithm from Claim 2.15 on the instance (H, Z, Z', τ, c^*) of the Heavy Vertex problem, where $Z = V(J) \cap V^{\text{in}}$; $Z' = V^{\text{out}} \setminus V(J)$; $c^* = \frac{M' \cdot \gamma}{2^j}$; $\tau = 2^j + 1$; and $W = W_{\max}$.
   (b) If the algorithm returns bit $b = 1$, a vertex $v^{\text{out}} \in Z'$ and vertices $u_1^{\text{in}}, \ldots, u_{\tau+1}^{\text{in}} \in Z$:
   (Observe that there is at most one vertex u_r^{in} such that edge $(u_r^{\text{in}}, v^{\text{out}})$ is special; assume w.l.o.g. that, if such a vertex u_r^{in} exists then $r = \tau + 1$.)
      - Add v^{out} to X and to X^*;
      - Add v^{out} to J, and, for all $1 \leq a \leq \tau$, add edge $(u_a^{\text{in}}, v^{\text{out}})$ to J;
      - Increase N' by 1, and, if $N' = N$ holds, halt.
      - go to Step (2)

Figure 3: Procedure Explore H'

After augmenting f with this flow, Property Q'1 will hold, since, for every vertex $u \in V(G)$ with $w(u) \geq M'$, we decreased the capacity of the special edge $(u^{\text{in}}, u^{\text{out}})$ in J' by at least M' units. In the remainder of the proof we show that, for every vertex $v \in V(J)$, there is an s^{out}-v flow in J' of value at least $\frac{M' \cdot \gamma}{2}$.

The proof is by induction over the time when the vertex v was added to J. The base of the induction is the beginning of the algorithm, when J only contains the vertex s^{out}, so the claim trivially holds.

Consider now some time $\mathcal{T}$ during Procedure Explore H', when a vertex v was added to J, and assume that the claim holds for all vertices that were added before v to J. Vertex v may only be added to J either during Step 2(c)i, or Step 2(c)ii, or Step 3b of the procedure. Assume first that v was added during Step 2(c)i. Then there is a vertex u that already lied in J before time $\mathcal{T}$, such that an edge (u, v) of capacity at least $M' \cdot \gamma$ is present in H, and is added to J at time $\mathcal{T}$. From the induction hypothesis, there is a flow f' in J', that sends $\frac{M' \cdot \gamma}{2}$ flow units from s^{out} to u. Since the capacity of edge (u, v) in J' is at least $M' \cdot \gamma - 2M' \geq \frac{M' \cdot \gamma}{2}$, we can extend this flow by sending $\frac{M' \cdot \gamma}{2}$ flow units along the edge (u, v), obtaining a flow in J', that sends $\frac{M' \cdot \gamma}{2}$ flow units from s^{out} to v.

Assume now that v was added to J during Step 2(c)ii or Step 3b. Then $v \in V^{\text{out}}$, and there is some integer $1 \leq j \leq \gamma$, and a collection Y of 2^j vertices of V^{in} that lied in J before time $\mathcal{T}$, such that, for every verex $u \in Y$, a regular edge (u, v) of capacity at least $\frac{M' \cdot \gamma}{2^j}$ was added to J at time $\mathcal{T}$. Denote $Y = \{u_1, \ldots, u_{2^j}\}$. From the induction hypothesis, for all $1 \leq r \leq 2^j$, there is a flow f_r of value $\frac{M' \cdot \gamma}{2}$ from s^{out} to u_r in graph J'. For all $1 \leq r \leq 2^j$, let $\mathcal{P}_r$ be a flow-path decomposition of f_r, and, for every path $P \in \mathcal{P}_r$, let $f_r(P)$ be the amount of flow sent via this path by f_r. Notice that, for all $1 \leq r \leq 2^j$, edge (u_r, v) is regular, and so its capacity in J' remains the same as in J, and is at least $\frac{M' \cdot \gamma}{2^j}$.

Let f' be the flow obtained by taking the union of the flows $f_1, \ldots, f_{2^j}$, scaled down by factor 2^j. In other words, for all $1 \leq r \leq 2^j$, for every path $P \in \mathcal{P}_r$, flow f' sends $\frac{f_r(P)}{2^j}$ flow units via the path P, simultaneously. Then it is easy to verify that f' is a valid flow in J' (that is, it respects the edge capacities), the total amount of flow that s^{out} sends via f' is $\frac{M' \cdot \gamma}{2}$, and, for all $1 \leq r \leq 2^j$, vertex u_r receives exactly $\frac{M' \cdot \gamma}{2^j}$ flow units. For all $1 \leq r \leq 2^j$, we send the $\frac{M' \cdot \gamma}{2^j}$ flow units that terminate at u_r, via the edge (u_r, v), to vertex v, obtaining a valid s^{out}-v flow in J', of value $\frac{M' \cdot \gamma}{2}$. We conclude that there exists an s^{out}-v flow in J' of value $\frac{M' \cdot \gamma}{2}$. □

Running Time Analysis. We now analyze the running time of Procedure Explore H'. We first analyze the running time while ignoring the time spent on calls to the algorithm from Claim 2.15 for the Heavy Vertex problem. Let Γ be the set of all vertices of $V^{\text{out}} \setminus \{s^{\text{out}}\}$ that have been added to J by Procedure Explore H', so $|\Gamma| \leq N$ holds. The time required to initialize the data structures is $O(n \log n)$. Notice that the running time of Procedure Explore H' (excluding the time spent on calls to the algorithm from Claim 2.15) can be asymptotically bounded by the total number of edges that the algorithm considered, times $O(\log n)$. Recall that every edge of G'' that is not incident to t has one of its endpoints in V^{out}. We partition the edges that the procedure explored into two subsets: set E^1 contains all edges that have an endpoint in Γ, and set E^2 contains all remaining edges. Clearly, $|E^1| \leq n \cdot |\Gamma| \leq n \cdot N \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W\max)}{\gamma}\right)$, since $N = \frac{32n^{1+2\epsilon} \cdot \log^2(W\max)}{\gamma}$. Next, we bound the number of edges of E^2 that the algorithm ever considered in Step 2(c)ii. Observe that, for every vertex $v^{\text{out}} \in V^{\text{out}}$ and integer $1 \leq j \leq j^*$, once 2^j edges from set E^2 that are incident to v^{out} are considered by the algorithm, vertex v^{out} is added to J and to Γ. Therefore, the total number of edges of E^2 considered by the algorithm in this step is bounded by:

$$O\left(n \cdot 2^{j^*}\right) \leq O\left(n \cdot \tau^*\right) \leq O\left(\frac{n^2}{\sqrt{\gamma}}\right),$$

since $j^* = \lceil \log(\tau^*) \rceil$, and $\tau^* = \frac{n}{\sqrt{\gamma}}$.

The time required to consider the remaining edges of E^2 is asymptotically bounded by the time spent on calls to the algorithm from Claim 2.15, that we bound next. Recall that the running time of the algorithm from Claim 2.15, on input (G, Z, Z', τ, c^*), is $O\left(\frac{n \cdot |Z| \cdot \log n \cdot \log(W_{\max})}{\tau}\right) \leq O\left(\frac{n^2 \cdot \log n \cdot \log(W_{\max})}{\tau}\right)$. Since we only apply the claim to values $\tau = 2^j$ where $j^* < j \leq \log \gamma$, where $j^* = \lceil \log(\tau^*) \rceil$ and $\tau^* = \frac{n}{\sqrt{\gamma}}$, we get that the running time of a single application of the algorithm from Claim 2.15 is:

$$O\left(\frac{n^2 \cdot \log n \cdot \log(W_{\max})}{\tau^*}\right) \leq O\left(n \cdot \sqrt{\gamma} \cdot \log n \cdot \log(W_{\max})\right).$$

We use the following observation to bound the number of calls to the algorithm from Claim 2.15 in a single iteration.

Observation 5.2 *The number of times that Procedure* Explore H' *executes the algorithm from Claim 2.15 is bounded by* $O(N \log n) \leq O\left(\frac{n^{1+2\epsilon} \cdot \log n \cdot \log^2(W_{\max})}{\gamma}\right)$.

Proof: Note that all calls to the algorithm from Claim 2.15 occur during Step 3, and every time Step 3 is executed, the algorithm from Claim 2.15 is called at most $O(\log n)$ times.

If Procedure Explore H' does not terminate once Step 3 is executed, then at least one vertex is added to Γ during this step. Therefore, Step 3 may be executed at most $O(N) \leq O\left(\frac{n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right)$ times. Altogether, the total number of calls to the algorithm from Claim 2.15 by Procedure Explore H' is bounded by $O(N \cdot \log n) \leq O\left(\frac{n^{1+2\epsilon} \cdot \log n \cdot \log^2(W_{\max})}{\gamma}\right)$. □

We conclude that the total running time that Procedure Explore H' spends on Step 3 is bounded by:

$$O\left(n \cdot \sqrt{\gamma} \cdot \log n \cdot \log(W_{\max})\right) \cdot O\left(\frac{n^{1+2\epsilon} \cdot \log n \cdot \log^2(W_{\max})}{\gamma}\right) \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2 n \cdot \log^3(W_{\max})}{\sqrt{\gamma}}\right).$$

To summarize, the running time of Procedure Explore H' is bounded by:

$$O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right) + \left(\frac{n^2}{\sqrt{\gamma}}\right) + O\left(\frac{n^{2+2\epsilon} \cdot \log^2 n \cdot \log^3(W_{\max})}{\sqrt{\gamma}}\right)$$
$$\leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2 n \cdot \log^3(W_{\max})}{\sqrt{\gamma}}\right).$$

Bounding $|E(J)|$. We use the following observation to bound $|E(J)|$.

Observation 5.3 $|E(J)| \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right)$.

Proof: As observed already, every edge of $E(J)$ must be incident to a vertex of $\Gamma \cup \{t\}$. Since $|\Gamma| \leq N$, we get that $|E(J)| \leq O(n \cdot N) \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right)$, since $N = \frac{32 n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}$. □

Once the algorithm from Procedure Explore H' terminates, we consider the following three cases.

Case 1: a vertex $x \in \hat{T}$ is returned. The first case happens if the algorithm for Procedure Explore H' terminates with a vertex $x \in \hat{T}$. Let $J' \subseteq J$ be the graph obtained from J as before, by reducing the capacity of every special edge in J by at least M' and at most $2M'$ units, so it becomes an integral multiple of M'. From Claim 5.1, there is a flow f' in J' that is M'-integral, in which s^{out} sends $\frac{M' \cdot \gamma}{2}$ flow units to x. We use the algorithm from Theorem 2.4 to compute this flow in time $O(|E(J)|^{1+o(1)} \cdot \log W_{\max})$. We extend this flow by sending $\frac{M' \cdot \gamma}{2}$ flow units via the edge (x, t) obtaining an s^{out}-t flow f'' in H of value $\frac{M' \cdot \gamma}{2}$, that is M'-integral. Finally, we augment the flow f with this resulting flow f'', and update the residual flow network H. It is easy to verify that the resulting flow f remains M'-integral, and that it has Property Q'1. The additional time that the algorithm spends in this case is bounded by $O(|E(J)|^{1+o(1)} \cdot \log(W_{\max}))$.

Case 2: $N' = N$. The second case happens if Procedure Explore H' terminated because $N' = N$ held, so, at the end of the procedure, $|\Gamma| = N$ holds. We select a single vertex $v^{\text{out}} \in \Gamma$ uniformly at random. We then perform a shortcut operation, by adding an edge (v^{out}, t) of capacity $W_{\max}$ to graph G'' and to the residual flow network H. The remainder of the algorithm for this case is identical to the algorithm for Case 1, except that vertex x is replaced with v^{out}. As before, from Claim 5.1, there is a flow f' in J' that is M'-integral, in which s^{out} sends $\frac{M' \cdot \gamma}{2}$ flow units to v^{out}. We use the algorithm from Theorem 2.4 to compute such a flow in time $O(|E(J)|^{1+o(1)} \cdot \log(W_{\max}))$. We extend this flow by sending $\frac{M' \cdot \gamma}{2}$ flow units via the edge (v^{out}, t), obtaining an s^{out}-t flow f'' in H of value $\frac{M' \cdot \gamma}{2}$, that is M'-integral. Finally, we augment the flow f with this resulting flow f'', and update the residual flow network H. It is easy to verify that the resulting flow f remains M'-integral, and that it has Property Q'1. The additional time that the algorithm spends in this case is bounded by $O(|E(J)|^{1+o(1)} \cdot \log(W_{\max}))$.

This completes the description of Case 2. We use the following simple claim to show that the probability that the resulting shortcut operation was invalid is low.

Claim 5.4 *If the algorithm chooses to perform a shortcut operation in an iteration, then the probability that this shortcut operation is invalid is at most* $\frac{\gamma}{32n^{1+\epsilon} \cdot \log^2(W_{\max})}$.

Proof: Let (v^{out}, t) be the shortcut edge that the algorithm has added. Note that the shortcut operation may only be invalid if (i) (s, T) is a good pair; and (ii) $v \in L$ holds. Since $|\Gamma| = N = \frac{32n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}$, and since, from Inequality 3, if (s, T) is a good pair, $|L| \leq n^{\epsilon}$ holds. We then get that, if (s, T) is a good pair, the probability that $v^{\text{out}} \in L$ is bounded by:

$$\mathbf{Pr}\left[v \in L\right] \leq \frac{|L|}{|\Gamma|} \leq \frac{\gamma}{32n^{1+\epsilon} \cdot \log^2(W_{\max})}.$$

□

Case 3. The third case happens when neither of the first two cases happen. In this case, the current iteration becomes the last one, and we construct two graphs, $\tilde{H}$ and $\tilde{H}'$, to be used in Step 2 of the algorithm, whose construction we describe later.

Running time of an iteration. From our discussion, the running time of a single iteration, excluding the time required to construct the graph $\tilde{H}$, is bounded by:

$$
\begin{aligned}
&O\left(\frac{n^{2+2\epsilon}\cdot\log^2 n\cdot\log^3(W_{\max})}{\sqrt{\gamma}}\right)+O(|E(J)|^{1+o(1)}\cdot\log W_{\max})\\
&\qquad\leq O\left(\frac{n^{2+2\epsilon}\cdot\log^2 n\cdot\log^3(W_{\max})}{\sqrt{\gamma}}\right)+O\left(\frac{n^{2+2\epsilon+o(1)}\cdot\log^4(W_{\max})}{\gamma}\right)\\
&\qquad\leq O\left(\frac{n^{2+2\epsilon+o(1)}\cdot\log^4(W_{\max})}{\sqrt{\gamma}}\right),
\end{aligned}
$$

from Observation 5.3.

5.2 Completing Step 1

The algorithm for Step 1 performs $z'=\left\lceil\frac{32n}{\gamma}\right\rceil$ iterations as described above. If, after z' iterations, the algorithm for a single iteration still terminates with cases 1 or 2, then we terminate the algorithm for the current phase with a "FAIL".

Note that the running time of the algorithm for Step 1 so far is:

$$
\begin{aligned}
&O\left(\frac{n^{2+2\epsilon+o(1)}\cdot\log^4(W_{\max})}{\sqrt{\gamma}}\right)\cdot z'\\
&\qquad\leq O\left(\frac{n^{3+2\epsilon+o(1)}\cdot\log^4(W_{\max})}{\gamma^{3/2}}\right)\\
&\qquad\leq O\left(n^{2-4\epsilon+o(1)}\cdot\log^4(W_{\max})\right),
\end{aligned}
$$

since $\gamma\geq n^{2/3+4\epsilon}$ from Inequality 7.

Let $\hat{\mathcal{E}}$ be the bad event that the algorithm from Claim 2.15 for the Heavy Vertex problem erred at any time when it was called during the execution of Step 1. Recall that the probability that the algorithm from Claim 2.15 errs in a single execution is at most $\frac{1}{n^{10}\cdot\log^6(W_{\max})}$, and, from Observation 5.2, the algorithm is executed at most $O\left(\frac{n^{1+2\epsilon}\cdot\log n\cdot\log^2(W_{\max})}{\gamma}\right)$ in every iteration. Since the number of iterations is bounded by $z'=\left\lceil\frac{32n}{\gamma}\right\rceil$, and since we assumed that n is sufficiently large, from the Union Bound, we get that:

$$
\mathbf{Pr}\left[\hat{\mathcal{E}}\right]\leq\frac{1}{n^{10}\cdot\log^6(W_{\max})}\cdot O\left(\frac{n^{2+2\epsilon}\cdot\log n\cdot\log^2(W_{\max})}{\gamma^2}\right)\leq\frac{1}{n^7\cdot\log^4(W_{\max})}. \tag{12}
$$

Let J be the graph constructed by Procedure Explore H' in the last iteration, and let $E^R\subseteq E(H)$ be the set of all **regular** edges $(v^{\text{in}},u^{\text{out}})$ with $v^{\text{in}}\in V(J)\cap V^{\text{in}}$ and $u^{\text{out}}\in V^{\text{out}}\setminus V(J)$. For all $1\leq j\leq\log\gamma$, let $E^R_j\subseteq E^R$ denote the set of all edges $e\in E^R$ with $\frac{M'\cdot\gamma}{2^j}\leq\hat{c}(e)<\frac{M'\cdot\gamma}{2^{j-1}}$, where $\hat{c}(e)$ is the capacity of edge e in graph H. We use the following claim that bounds the number of regular edges of H whose first endpoint lies in J.

Claim 5.5 *If $e=(x,y)\in E(H)$ is a regular edge with $x\in V(J)$ and $y\notin V(J)$, then $e\in E^R$. Additionally, $E^R=\bigcup_{j=1}^{\log\gamma}E^R_j$ holds. Lastly, if Event $\hat{\mathcal{E}}$ did not happen, then, for all $1\leq j\leq\log\gamma$,*

every vertex $u^{\text{out}} \in V^{\text{out}} \setminus V(J)$ is incident to at most $2^{j+11} \cdot \log n \cdot \log(W_{\max})$ edges of E_j^R, and at most $O(\gamma \cdot \log n \cdot \log(W_{\max}))$ edges of E^R.

Proof: Let $e = (x, y) \in E(H)$ be a regular edge with $x \in V(J)$ and $y \notin V(J)$. Assume first that $x = v^{\text{out}}$ and $y = u^{\text{in}}$, for some $v^{\text{out}} \in V^{\text{out}}$ and $u^{\text{in}} \in V^{\text{in}}$. Then edge (v, u) lies in G, and so the regular edge $(v^{\text{out}}, u^{\text{in}})$ of capacity $W_{\max}$ was added to the split graph G'. Since the flow on this edge is bounded by $\max_{v \in V(G)} \{w(v)\} \leq W'_{\max} < W_{\max}/2$, we get that the capacity of the edge e in H must remain at least $W_{\max}/2 \geq M' \cdot \gamma$. Therefore, Procedure Explore H' should have added u^{in} to J in Step 2(c)i, a contradiction. It then must be the case that $x = v^{\text{in}}$ and $y = u^{\text{out}}$ for some $v^{\text{in}} \in V^{\text{in}}$ and $u^{\text{out}} \in V^{\text{out}}$, so $e \in E^R$.

Next, we prove that $E^R = \bigcup_{j=1}^{\log \gamma} E_j^R$. Consider any regular edge $e = (v^{\text{in}}, u^{\text{out}}) \in E^R$. Note that it is impossible that $\hat{c}(e) \geq M' \cdot \gamma$, since then e and u^{out} should have been included in J when Procedure Explore H' was applied for the last time. We now claim that $\hat{c}(e) \geq M'$ must also hold. Indeed, assume otherwise. Notice that edge $e = (v^{\text{in}}, u^{\text{out}})$ does not exist in graph G'', from the definition of the split graph. If $\hat{c}(e) < M'$, then the current flow on edge $(u^{\text{out}}, v^{\text{in}})$ is non-zero, but below M'. Since the current flow is M'-integral, this is impossible. We conclude that $E^R = \bigcup_{j=1}^{\log \gamma} E_j^R$.

Consider now an integer $1 \leq j \leq \gamma$, and the set E_j^R of edges; recall that it contains all edges $e \in E^R$ with $\frac{M' \cdot \gamma}{2^j} \leq \hat{c}(e) < \frac{M' \cdot \gamma}{2^{j-1}}$. Assume first that $j \leq j^*$. If some vertex $u^{\text{out}} \in V^{\text{out}} \setminus V(J)$ is incident to more than 2^j edges of E_j^R, then Step 2(c)ii of Procedure Explore H' should have discovered it, when the procedure was executed for the last time, and u^{out} must have been added to J. Therefore, the number of edges of E_j^R incident to any vertex $u^{\text{out}} \in V^{\text{out}} \setminus V(J)$ for $j \leq j^*$ is bounded by 2^j.

Lastly, consider an integer $j^* < j \leq \gamma$, and assume for contradiction that there is a vertex $u^{\text{out}} \in V^{\text{out}} \setminus V(J)$ that is incident to more than $2^{j+11} \cdot \log n \cdot \log(W_{\max})$ edges of E_j^R. Consider the last time when Step 3b of Procedure Explore H' was executed during the last iteration, and the call to the algorithm from Claim 2.15 for the Heavy Vertex problem with parameters $c^* = \frac{M' \cdot \gamma}{2^j}$ and $\tau = 2^j + 1$ during the execution of that step. From our assumption, there are at least $2^{j+11} \cdot \log n \cdot \log(W_{\max}) > 1000\tau \log n \log(W_{\max})$ edges of capacity at least c^* connecting vertices of $Z = V(J) \cap V^{\text{in}}$ to the vertex u^{out}, that lies in Z'. Since this is the last time when Step 2(c)ii was executed, the algorithm from Claim 2.15 must have returned bit $b = 0$, and so it must have erred, contradicting the assumption that Event $\hat{\mathcal{E}}$ did not happen. □

As our last step, we verify that, for all $1 \leq j \leq \log \gamma$, $|E_j^R| \leq |V^{\text{out}} \setminus V(J)| \cdot 2^{j+11} \cdot \log n \cdot \log(W_{\max})$. If we discover that this is not the case for any such value j, then we terminate the algorithm for the current phase with a "FAIL". Notice that, from Claim 5.5, this may only happen if Event $\hat{\mathcal{E}}$ occurred.

In order to execute this last step efficiently, for all $1 \leq j \leq \log \gamma$, we maintain a counter N_j, that counts the number of edges in E_j^R; we initialize all these counters to 0. We then consider every vertex $u^{\text{in}} \in V^{\text{in}} \cap V(J)$ in turn. When such a vertex u^{in} is considered, we use the modified adjacency-list representation of graph H in order to count all regular edges $e = (u^{\text{in}}, v^{\text{out}})$ leaving u^{in}, that lie in E_j^R, for all $1 \leq j \leq \log \gamma$, and update the counter N_j accordingly. Once the counter N_j for any such integer $1 \leq j \leq \log \gamma$ becomes greater than $|V^{\text{out}} \setminus V(J)| \cdot 2^{j+11} \cdot \log n \cdot \log(W_{\max})$, we terminate the algorithm with a "FAIL". Note that, for every vertex $u^{\text{in}} \in V^{\text{in}} \cap V(J)$, for every regular edge $e = (u^{\text{in}}, v^{\text{out}})$ whose capacity is at least M', if $e \notin E^R$, then $v^{\text{out}} \in \Gamma$ holds. Therefore, the time required to execute this step is bounded by:

$$O(N \cdot n) + O(n\gamma \log n \cdot \log^2(W_{\max})) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^2(W_{\max})\right),$$

since $N = \frac{32n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}$ and $n^{6\epsilon} \leq \gamma \leq n^{1-4\epsilon}$ from Inequality 7.

Construction of Graphs $\tilde{H}'$ and $\tilde{H}$. At the end of Step 1 we construct the graph $\tilde{H}'$, that is defined as follows. Graph $\tilde{H}'$ is obtained from H, by contracting all vertices of $V(H) \setminus V(J)$ into a supersink t', while keeping the parallel edges, but deleting all edges that leave t', including self-loops. For every edge $e = (x, y)$ of H with $x \in V(J)$ and $y \notin V(J)$, there is a unique edge $e' = (x, t)$ corresponding to e in $\tilde{H}'$, whose capacity in $\tilde{H}'$ is equal to the capacity of e in H. We sometimes say that e' *represents* e in $\tilde{H}'$, or that e' is a *copy* of e in $\tilde{H}'$, and we may not distinguish between e and e'.

From Claim 5.5, it is easy to verify that every edge $e \in E(\tilde{H}')$ belongs to one of the following three cathegories:

- either one endpoint of e lies in Γ;
- or e is a copy of an edge $e' \in E^R$;
- or e is a copy of a special edge $e' = (v^{\text{in}}, v^{\text{out}})$, with $v^{\text{in}} \in V^{\text{in}} \cap V(J)$.

Notice that the number of all edges incident to the vertices of Γ is bounded by $|\Gamma| \cdot n \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right)$, and the set of all such edges can be constructed in time $O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^2(W_{\max})\right)$ by simply inspecting all edges that are incident to the vertices of Γ using the adjacency-list representation of H. The edges of E^R, and all special edges $e = (v^{\text{in}}, v^{\text{out}})$ with $v^{\text{in}} \in V^{\text{in}} \cap V(J)$ can be computed in time $O(n\gamma \log n \cdot \log^2(W_{\max})) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^2(W_{\max})\right)$ using the procedure described above for verifying that $|E_j^R|$ is not too large for all $1 \leq j \leq \log \gamma$. Altogether, graph $\tilde{H}'$ can be constructed in time:

$$O\left(n^{2-4\epsilon+o(1)} \cdot \log^2(W_{\max})\right).$$

Finally, we construct the graph $\tilde{H}$, that is identical to $\tilde{H}'$, except that we unify parallel edges into a single edge, whose capacity is equal to the total capacity of the corresponding parallel edges.

Altogether, the total running time of Step 1 then remains bounded by $O\left(n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right)$. We summarize the properties of graph $\tilde{H}'$ that will be useful for us in the next step, in the following observation.

Observation 5.6 *If the algorithm for Step 1 did not terminate with a "FAIL", then the graph $\tilde{H}'$ that it constructed has the following properties:*

- *P1. The total number of edges $e \in E(\tilde{H}')$ that have one endpoint in $V(J) \cap V^{\text{out}}$ and another in $V(J) \setminus V^{\text{out}}$ is bounded by $O(n \cdot N) \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right)$.*
- *P2. Every other edge of $E(\tilde{H}')$ that is not special, is of the form (v^{in}, t'), where $v^{\text{in}} \in V(J) \cap V^{\text{in}}$, and corresponds to an edge of E^R, that is, a regular edge $e = (v^{\text{in}}, u^{\text{out}}) \in E(H)$, where $u^{\text{out}} \in V^{\text{out}} \setminus V(J)$, $u \neq v$, and $M' \leq \hat{c}(e) < M' \cdot \gamma$, where $\hat{c}(e)$ is the capacity of e in H.*
- *P3. If we denote, for all $1 \leq j \leq \log \gamma$, by E_j^R the set of all edges $e \in E^R$ with $\frac{M' \cdot \gamma}{2^j} \leq \hat{c}(e) < \frac{M' \cdot \gamma}{2^{j-1}}$, then $|E_j^R| \leq n \cdot 2^{j+11} \cdot \log n \cdot \log(W_{\max})$ holds. Moreover, if bad event $\hat{\mathcal{E}}$ did not happen, then, for all $1 \leq j \leq \log \gamma$, every vertex $v^{\text{out}} \in V^{\text{out}} \setminus V(J)$ is incident to at most $2^{j+11} \log n \cdot \log(W_{\max})$ edges of E_j^R.*

P4. Altogether, $|E^R| \leq O\left(n \cdot \gamma \cdot \log n \cdot \log(W_{\max})\right)$*, and the total capacity of the edges of* E^R *in* H *is at most* $2^{12} n \cdot M' \cdot \gamma \cdot \log^2 n \log(W_{\max})$*. Also:*

$$|E(\hat{H}')| \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right) + O\left(n \cdot \gamma \cdot \log n \cdot \log(W_{\max})\right) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^2(W_{\max})\right).$$

We also obtain the following useful corollary of the observation:

Corollary 5.7 *Assume that* (s,T) *is a good pair, that all shortcut operations so far have been valid, and that Event* $\hat{\mathcal{E}}$ *did not happen. Let* A^* *be the set of all edges* $(x,y) \in E(H)$*, where* $x \in V(J)$ *and* $y \in L^* \setminus V(J)$*. Then* $\sum_{e \in A^*} \hat{c}(e) \leq 2^{14} \cdot n^{\epsilon} \cdot \gamma \cdot M' \cdot \log n \cdot \log(W_{\max})$.

Proof: We partition the set A^* of edges into two subsets: set A_1 contains all edges $(x,y) \in A^*$ with $y \in L^{\text{in}}$, and A_2 contains all remaining edges.

Consider first some edge $e = (x,y) \in A_1$. Since $y \in L^{\text{in}}$, from the construction of the split graph G', it must be the case that $x \in V^{\text{out}}$. We denote $x = v^{\text{out}}$ and $y = u^{\text{in}}$, where $u \in L$ and $v \in V(G)$. From Claim 5.5, e may not be a regular edge. Therefore, e must be a special edge, $e = (v^{\text{out}}, v^{\text{in}})$, for some vertex $v \in L$. Since v^{in} was not added to J during the last application of Procedure Explore H', we get that $\hat{c}(e) < M' \cdot \gamma$. Therefore, $\sum_{e \in A_1} \hat{c}(e) \leq |L| \cdot M' \cdot \gamma$.

Next, we bound the total capacity of the edges in A_2. For every vertex $v^{\text{out}} \in L^{\text{out}} \setminus V(J)$, let $A_2(v^{\text{out}}) \subseteq A_2$ denote the subset of all edges $e \in A_2$ that are incident to v^{out}. From Property P3 of Observation 5.6, for all $1 \leq j \leq \gamma$, v^{out} may be incident to at most $2^{j+11} \cdot \log n \cdot \log(W_{\max})$ edges of E^R_j: regular edges (a, v^{out}) with $a \in V(J)$, whose capacity is at most $\frac{M'\gamma}{2^{j-1}}$. Altogether, the total capacity of all regular edges $(a, v^{\text{out}}) \in A_2(v^{\text{out}})$ is at most $2^{12} \cdot M' \cdot \gamma \cdot \log n \cdot \log(W_{\max})$. Only one special edge (b, v^{out}) with $b \in V(J)$ may be incident to v^{out} in H, that must be a forward edge. Since vertex v^{out} was not added to J when Procedure Explore H' was executed for the last time, the capacity of this edge is bounded by $M'\gamma$. Overall, we get that $\sum_{e \in A_2(v^{\text{out}})} \hat{c}(e) \leq 2^{13} \cdot \gamma \cdot M' \cdot \log n \cdot \log(W_{\max})$, and $\sum_{e \in A_2} \hat{c}(e) \leq |L| \cdot 2^{13}\gamma \cdot M' \cdot \log n \cdot \log(W_{\max})$.

Since, from Inequality 3, $|L| \leq n^{\epsilon}$, we get that: $\sum_{e \in A_1 \cup A_2} \hat{c}(e) \leq 2^{14} \cdot n^{\epsilon} \cdot \gamma \cdot M' \cdot \log n \cdot \log(W_{\max})$. □

The next claim bounds the probability that one of the shortcut steps executed in Step 1 was invalid, or that the algorithm for this step terminated with a "FAIL".

Claim 5.8 *The probability that any shortcut operation executed in the current step is invalid is at most* $\frac{1}{n^{\epsilon} \cdot \log^2(W_{\max})}$*. Moreover, if* (s,T) *is a good pair, and if all shortcut operations performed before Phase* i *were valid, then the probability that the algorithm terminates with a "FAIL" in the first step is at most* $\frac{2}{n^{\epsilon} \cdot \log^2(W_{\max})}$.

Proof: Since the number of iterations is at most $z' = \frac{32n}{\gamma}$, and since, from Claim 5.4, for every iteration, the probability that the algorithm performs an invalid shortcut operation over the course of the iteration is at most $\frac{\gamma}{32n^{1+\epsilon} \cdot \log^2(W_{\max})}$, we get that overall, the probability that any shortcut operation executed in Step 1 is invalid is bounded by:

$$\frac{\gamma}{32n^{1+\epsilon} \cdot \log^2(W_{\max})} \cdot z' \leq \frac{1}{n^{\epsilon} \cdot \log^2(W_{\max})}.$$

Assume now that (s,T) is a good pair, and that all shortcut operations performed before Phase i were valid. We claim that, in this case, if the algorithm for Step 1 terminates with a "FAIL", then

either Event $\hat{\mathcal{E}}$ has happened, or at least one shortcut operation performed in the current step was invalid. Indeed, assume otherwise, so all shortcut operations executed by the algorithm were valid, and Event $\hat{\mathcal{E}}$ did not happen. The only way that the algorithm for Step 1 terminates with a "FAIL" in this case is if the algorithm did not terminate after z' iterations.

Observe that, since we assumed that all shortcut operations were valid, the value of the maximum s^{out}-t flow in the graph G'' remains equal to OPT_s throughout this step. From Invariant I3, at the beginning of the phase, we were given a flow f_{i-1} with $\text{val}(f_{i-1}) \geq \mathsf{OPT}_s - 4n \cdot M_{i-1} = \mathsf{OPT}_s - 8n \cdot M'$. Our algorithm performed $z' = \frac{32n}{\gamma}$ iterations, and in each iteration it augmented the flow by $\frac{M' \cdot \gamma}{2}$ flow units. Therefore, at the end of Step 1, we obtain a valid s^{out}-t flow in G'', whose value is:

$$\text{val}(f_{i-1}) + z' \cdot \frac{M' \cdot \gamma}{2} > \text{val}(f_{i-1}) + 8n \cdot M' \geq \mathsf{OPT}_s,$$

a contradiction. Therefore, if (s, T) is a good pair, and if all shortcut operations performed before Phase i were valid, the algorithm for Step 1 may only terminate with a "FAIL" if at least one of the shortcut operations it performed was invalid, or Event $\hat{\mathcal{E}}$ has happened. The probability of the former is bounded by $\frac{1}{n^{\epsilon} \cdot \log^2(W_{\max})}$, and, from Inequality 12, $\mathbf{Pr}\left[\hat{\mathcal{E}}\right] \leq \frac{1}{n^7 \cdot \log^4(W_{\max})}$. Altogether, we get that, if all shortcut operations performed before Phase i were valid, then the probability that the algorithm for Step 1 terminates with a "FAIL" is at most:

$$\frac{1}{n^{\epsilon} \cdot \log^2(W_{\max})} + \frac{1}{n^7 \cdot \log^4(W_{\max})} \leq \frac{2}{n^{\epsilon} \cdot \log^2(W_{\max})}.$$

□

6 Step 2: Computing the Partition (Q, Q') of U

In this step, we will use the graphs $\tilde{H}$ and $\tilde{H}'$ that were constructed in the previous step. The goal of this step is to compute a partition (Q, Q') of the set U of vertices, such that such that Q' only contains relatively few vertices of $R \cup L$. Additionally, if $|Q| > n^{1-4\epsilon}$, then we will also compute an s^{out}-t cut $(\hat{X}, \hat{Y})$ in H, such that, in any maximum s^{out}-t flow in H, the total flow over the edges in $E_H(\hat{X}, \hat{Y})$ is close to their capacity, and for every vertex $v \in Q$, $v^{\text{in}} \in \hat{X}$ and $v^{\text{out}} \in \hat{Y}$ holds. Recall that, for every edge $e \in E(H)$, we denoted by $\hat{c}(e)$ its capacity in H. Throughout this step, we denote by F the value of the maximum s^{out}-t flow in H, and by F' the value of the maximum s^{out}-t' flow in $\tilde{H}'$. We start by exploring some properties of the graphs H and $\tilde{H}'$.

6.1 Properties of Maximum s^{out}-t Flow in H

We define several subsets of edges of H, and then express the value F of the maximum s^{out}-t flow in H in terms of their capacities. We use the following definition:

Definition 6.1 (Set E'_S of edges.) *Consider the following three subsets of edges of H: set*

$$E^0_S = \left\{(v^{\text{in}}, v^{\text{out}}) \mid v \in S\right\}$$

that contains all forward special edges representing the vertices of S; set

$$E^1_S = \left\{(v^{\text{in}}, u^{\text{out}}) \mid v \in L \cup S, u \in S \setminus \{v\}\right\}$$

that contains all regular edges connecting in-copies of vertices of $L \cup S$ to out-copies of vertices of S; and set

$$E^2_S = \left\{(v^{\text{in}}, u^{\text{out}}) \mid v \in S, u \in R\right\}$$

that contains all regular edges connecting in-copies of vertices of S to out-copies of vertices of R. We set $E'_S = E^0_S \cup E^1_S \cup E^2_S$, and we denote by $\hat{c}_S = \sum_{e \in E'_S} \hat{c}(e)$ the total capacity of the edges of E'_S in H.

Note that every edge of E'_S has an endpoint in $S^{\text{out}} \cup R^*$.

We start with the following simple observation regarding the structure of H.

Observation 6.2 *Assume that (s,T) is a good pair, and that all shortcut operations so far have been valid. Consider the partition (X,Y) of the vertices of G'' (and of H), where $X = L^* \cup S^{\text{in}}$, and $Y = V(G'') \setminus X = S^{\text{out}} \cup R^* \cup \{t\}$. Then $E_H(X,Y) = E'_S$.*

Proof: It is immediate to verify that $E'_S \subseteq E_H(X,Y)$. We now prove that $E_H(X,Y) \subseteq E'_S$.

Consider any edge $e = (x,y)$ of H with $x \in X$ and $y \in Y$. Since (L,S,R) is a vertex cut in G, and since we have assumed that (s,T) is a good pair, and all shortcut operations so far had been valid, $y \neq t$, and moreover, either $x \notin L^*$, or $y \notin R^*$ holds.

Assume first that $x \notin L^*$, so $x = v^{\text{in}}$, for some $v \in S$. If e is a special edge, then $e = (v^{\text{in}}, v^{\text{out}}) \in E^0_S$ must hold. If e is a regular edge, then $e = (v^{\text{in}}, u^{\text{out}})$, where $u \in (S \cup R) \setminus \{v\}$ must hold, so $e \in E^1_S \cup E^2_S$.

Assume now that $y \notin R^*$, so $y = v^{\text{out}}$ for some $v \in S$. As before, if e is a special edge, then $e = (v^{\text{in}}, v^{\text{out}}) \in E^0_S$ must hold. If e is a regular edge, then $e = (u^{\text{in}}, v^{\text{out}})$, where $u \in S \cup L$ must hold, and $u \neq v$, so $e \in E^1_S$. In any case, $e \in E'_S$ must hold. □

Note that, from Observation 6.2, if (s,T) is a good pair, and if all shortcut operations so far have been valid, then graph $H \setminus E'_S$ contains no s^{out}-t path. Next, we express the value F of the maximum s^{out}-t flow in H, in terms of $\hat{c}_S$.

Claim 6.3 *Assume that (s,T) is a good pair, and that all shortcut operations so far have been valid. Then $F = \hat{c}_S$ and $F \leq 8n \cdot M'$ must hold. Moreover, if f' is a maximum s^{out}-t flow in H, and $\mathcal{P}$ is its flow-path decomposition, then every path in $\mathcal{P}$ contains exactly one edge of E'_S.*

Proof: Assume that (s,T) is a good pair, and that all shortcut operations so far have been valid. Let (X,Y) be the partition of $V(G'')$ that is defined as before: $X = L^* \cup S^{\text{in}}$, and $Y = V(G'') \setminus X = S^{\text{out}} \cup R^* \cup \{t\}$.

Consider the current s^{out}-t flow f in G'', and recall that its value is equal to the total amount of flow leaving X, minus the total amount of flow entering X, so $\text{val}(f) = f^{\text{out}}(X) - f^{\text{in}}(X)$, where $f^{\text{out}}(X) = \sum_{e \in E_{G''}(X,Y)} f(e)$ and $f^{\text{in}}(X) = \sum_{e \in E_{G''}(Y,X)} f(e)$. From the definition of the split graph G'', and since (L,S,R) is a vertex cut in G, it is immediate to verify that $E_{G''}(X,Y) = \left\{(v^{\text{in}}, v^{\text{out}}) \mid v \in S\right\} = E^0_S$ (we have used the fact that, from Property Q'1, the residual capacity of each original special edge of G'' in H is greater than 0). Therefore, $f^{\text{out}}(X) = \sum_{e \in E^0_S} f(e)$. It is easy to verify that, if $e = (x,y) \in E_{G''}(Y,X)$, then e is a regular edge, where $x = v^{\text{out}}$ for some vertex $v^{\text{out}} \in V^{\text{out}}$, and $y = u^{\text{in}}$ for some vertex $u^{\text{in}} \in V^{\text{in}} \setminus \left\{v^{\text{in}}\right\}$; moreover, either (i) $v \in R \cup S$ and $u \in S$; or (ii) $v \in S$ and $u \in L \cup S$ must hold. In other words, each such edge must be anti-parallel to some edge in $E^1_S \cup E^2_S$. From the definition of the split graph G'', the edges of $E^1_S \cup E^2_S$ may not lie in G''. So for every edge $(u^{\text{in}}, v^{\text{out}}) \in E^1_S \cup E^2_S$, its residual capacity in H is: $\hat{c}(u^{\text{in}}, v^{\text{out}}) = f(v^{\text{out}}, u^{\text{in}})$. Therefore:

$$f^{\text{in}}(X) = \sum_{e \in E_S^1 \cup E_S^2} \hat{c}(e).$$

Altogether, we get that:

$$\text{val}(f) = f^{\text{out}}(X) - f^{\text{in}}(X) = \sum_{e \in E_S^0} f(e) - \sum_{e \in E_S^1 \cup E_S^2} \hat{c}(e).$$

Recall that the value of the maximum s^{out}-t flow in G'' is $\mathsf{OPT}_s = \sum_{e \in E_S^0} c(e)$, and the value of the maximum s^{out}-t flow in H is:

$$\begin{aligned} F &= \mathsf{OPT}_s - \text{val}(f) \\ &= \sum_{e \in E_S^0} c(e) - \sum_{e \in E_S^0} f(e) + \sum_{e \in E_S^1 \cup E_S^2} \hat{c}(e) \\ &= \sum_{e \in E_S^0} \hat{c}(e) + \sum_{e \in E_S^1 \cup E_S^2} \hat{c}(e) \\ &= \hat{c}_S. \end{aligned}$$

Consider now the maximum s^{out}-t flow f' in H, whose value is $F = \hat{c}_S = \sum_{e \in E'_S} \hat{c}(e)$, and let $\mathcal{P}$ be its flow-path decomposition. Since, from Observation 6.2, the edges of E'_S separate s^{out} from t in H, we get that every flow-path $P \in \mathcal{P}$ must contain at least one edge of E'_S. Since the total flow on the paths in $\mathcal{P}$ is equal to the total capacity of the edges of E'_S, we get that every path in $\mathcal{P}$ must contain exactly one edge of E'_S.

It remains to show that $F \leq 8n \cdot M'$. Recall that, if (s, T) is a good pair, and all shortcut operations so far have been valid, then, from Claim 4.1, the value of the maximum s^{out}-t flow in G' is OPT_s, and so the value of the maximum s^{out}-t flow in the residual flow network H is equal to $\mathsf{OPT}_s - \text{val}(f)$. Since flow f was obtained by augmenting the flow f_{i-1}, from Invariant I3, $\text{val}(f) \geq \text{val}(f_{i-1}) > \mathsf{OPT}_s - 4nM_{i-1} = \mathsf{OPT}_s - 8nM'$, since $M' = M_i = M_{i-1}/2$. Therefore, $F = \mathsf{OPT}_s - \text{val}(f) \leq 8nM'$. □

6.2 Properties of the Maximum s^{out}–t' Flow in $\tilde{H}'$

Next, we explore the properties of the maximum s^{out}-t' flow in $\tilde{H}'$, whose value is denoted by F'. We start by defining and analyzing a special type of s^{out}-t' paths in $\tilde{H}'$, that we call *problematic* paths.

Definition 6.4 (Problematic path) *Consider any s^{out}-t' path P in $\tilde{H}'$, and denote its last edge by $e = (x, t')$. Let $e' = (x, y) \in E(H)$ be the edge of H that edge e represents. We say that path P is* problematic, *if $E(P) \cap E'_S = \emptyset$, and $e' \notin E'_S$.*

We will use the following observation in order to bound the total flow carried by problematic paths in any s^{out}-t' flow in $\tilde{H}'$.

Observation 6.5 *Assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let P be any s^{out}-t' path in $\tilde{H}'$ that is problematic, let $e = (x, t')$ be the last edge on P, and let $e' = (x, y)$ be the edge of H that edge e represents. Then either (i) $x = v^{\text{out}}$*

and $y = v^{\text{in}}$ for some vertex $v \in L$, so e' is a backward special edge representing a vertex in L; or (ii) $y \in L^{\text{out}}$.

Proof: Let $P' \subseteq H$ be the path obtained from P by replacing its last edge with edge $e' = (x, y)$. Since P' contains no edges of E'_S, and its first vertex is $s^{\text{out}} \in L^{\text{out}}$, from Observation 6.2, all vertices on P' must lie in $L^* \cup S^{\text{in}}$.

Assume first that e' is a special edge. If e' a backward special edge, that is, $e' = (v^{\text{out}}, v^{\text{in}})$ for some vertex $v \in V(G)$, then, since all vertices of P' are contained in $L^* \cup S^{\text{in}}$, it must be the case that $v^{\text{out}} \in L^{\text{out}}$, and so $v \in L$.

If e' is a forward special edge, that is, $e' = (v^{\text{in}}, v^{\text{out}})$ for some vertex v, then, since $e' \notin E'_S$, $v \in L$ and $v^{\text{out}} \in L^{\text{out}}$ must hold.

Lastly, assume that e' is a regular edge. Then from Property P2 of Observation 5.6, $x = v^{\text{in}}$ for some vertex $v^{\text{in}} \in V^{\text{in}}$ and $y = u^{\text{out}}$ for some vertex $u^{\text{out}} \in V^{\text{out}}$. Since $y \in L^* \cup S^{\text{in}}$, we get that $y \in L^{\text{out}}$ must hold. □

We now obtain the following immediate corollary of the observation.

Corollary 6.6 *Assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let f'' be any s^{out}-t' flow in $\tilde{H}'$, let $\mathcal{P}$ be a flow-path decomposition of f'', and let $\mathcal{P}' \subseteq \mathcal{P}$ be the set of problematic paths. Then $\sum_{P \in \mathcal{P}'} f''(P) \leq 2^{14} n^{\epsilon} \cdot M' \cdot \gamma \cdot \log^2 n \cdot \log(W_{\max})$.*

Proof: We assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let A^* be the set of all edges $(x, y) \in E(H)$, where $x \in V(J)$ and $y \in L^* \setminus V(J)$. From Corollary 5.7, $\sum_{e \in A^*} \hat{c}(e) \leq 2^{14} \cdot n^{\epsilon} \cdot \gamma \cdot M' \cdot \log^2 n \cdot \log(W_{\max})$.

Consider now any s^{out}-t' flow f'' in $\tilde{H}'$, and let $\mathcal{P}$ be a flow-path decomposition of f''. Let $\mathcal{P}' \subseteq \mathcal{P}$ be the set of all problematic paths. Then from Observation 6.5, for every path $P \in \mathcal{P}'$, the last edge on P is a copy of an edge in A^*, and so $\sum_{P \in \mathcal{P}'} f''(P) \leq \sum_{e \in A^*} \hat{c}(e) \leq 2^{14} \cdot \gamma \cdot M' \cdot n^{\epsilon} \cdot \log^2 n \cdot \log(W_{\max})$. □

For brevity, we denote by $\eta = 2^{14} \cdot \gamma \cdot M' \cdot n^{\epsilon} \cdot \log^2 n \cdot \log(W_{\max})$. Recall that we denoted by $U \subseteq V(G)$ the set of all vertices $v \in V(G)$, for which $w(v) \geq M'$ holds. Let Q'_0 denote the set of all vertices $v \in U$ with $v^{\text{in}} \notin V(J)$. The next claim bounds the value F' of the maximum s^{out}–t' flow in $\tilde{H}'$ in terms of F, and also shows that only a small number of vertices of Q'_0 may lie in $S \cup L$, provided that (s, T) is a good pair, all shortcut operations so far have been valid, and bad event $\hat{\mathcal{E}}$ did not happen.

Claim 6.7 *Assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Then $F \leq F' \leq F + \eta$, and $|Q'_0 \cap (S \cup L)| \leq \frac{2\eta}{M'} \leq n^{1-8\epsilon}$.*

Proof: Recall that graph $\tilde{H}'$ is obtained from H by contracting all vertices of $V(H) \setminus V(J)$ into the sink t', and recall that $t \notin V(J)$. It is then immediate to see that any s^{out}-t flow in H defines an s^{out}-t' flow in $\tilde{H}'$ of the same value, so $F' \geq F$ must hold.

Next, we show that $|Q'_0 \cap (S \cup L)| \leq \frac{2\eta}{M'} \leq n^{1-8\epsilon}$. Consider an optimal s^{out}-t flow f' in H, whose value is F, and let $\mathcal{P}$ be a flow-path decomposition of this flow. Let $E''_S = \left\{(v^{\text{in}}, v^{\text{out}}) \mid v \in Q'_0 \cap S\right\}$ be the set of all forward special edges of H that correspond to the vertices of $Q'_0 \cap S$, so $E''_S \subseteq E^0_S$. For every edge $e \in E''_S$, let $\mathcal{P}(e) \subseteq \mathcal{P}$ be the set of flow-paths containing the edge e. From Claim 6.3, $F = \hat{c}_S$, and every flow-path in $\mathcal{P}$ contains exactly one edge of E'_S. Therefore, for every edge $e \in E''_S$, $\sum_{P \in \mathcal{P}(e)} f'(P) = \hat{c}(e)$.

Flow f' naturally defines an s^{out}-t' flow $\hat{f}$ of value F in graph $\tilde{H}'$, as follows. For every path $P \in \mathcal{P}$, let a_P be the first vertex on P that does not belong to $V(J)$, and let P' be obtained by taking the subpath of P from s^{out} to a_P, and replacing a_P with t'. We then set the flow $\hat{f}(P') = f'(P)$. It is immediate to verify that $\hat{f}$ is a valid s^{out}–t' flow of value F in graph $\tilde{H}'$. Consider now any edge $e = (v^{\text{in}}, v^{\text{out}}) \in E''_S$, and a path $P \in \mathcal{P}(e)$. Since $v^{\text{in}} \notin V(J)$, the corresponding path P' must be problematic. From Corollary 6.6:

$$\sum_{e \in E''_S} \sum_{P \in \mathcal{P}(e)} \hat{f}(P') \leq 2^{14} n^{\epsilon} \cdot M' \cdot \gamma \cdot \log^2 n \cdot \log(W_{\max}) = \eta.$$

On the other hand, since, from Property Q'1, the capacity of every edge $e \in E''_S$ is at least M', we get that:

$$\sum_{e \in E''_S} \sum_{P \in \mathcal{P}(e)} \hat{f}(P') = \sum_{e \in E''_S} \sum_{P \in \mathcal{P}(e)} f''(P) = \sum_{e \in E''_S} \hat{c}(e) \geq M' \cdot |E''_S| = M' \cdot |Q'_0 \cap S|.$$

Altogether, we get that:

$$|Q'_0 \cap (S \cup L)| \leq |Q'_0 \cap S| + |L| \leq \frac{\eta}{M'} + n^{\epsilon} \leq \frac{2\eta}{M'} \leq n^{1-8\epsilon},$$

from Inequality 3. We have also used the fact that $\eta = 2^{14} n^{\epsilon} \cdot M' \cdot \gamma \cdot \log^2 n \cdot \log(W_{\max})$, $\gamma \leq 2n^{1-10\epsilon}$ from Inequality 7, and $n^{\epsilon} > 2^{20} \cdot \log^2 n \cdot \log(W_{\max})$ from Inequality 5.

Consider now a maximum s^{out}-t' flow f'' in $\tilde{H}'$, whose value is F', and let $\mathcal{P}'$ be a flow-path decomposition of this flow. We partition the flow-paths in $\mathcal{P}'$ into two subsets: set $\mathcal{P}'_1$ containing all paths P that are non-problematic, and set $\mathcal{P}'_2$ containing all problematic paths. Since every path of $\mathcal{P}'_1$ contains an edge of E'_S (or an edge representing it), $\sum_{P \in \mathcal{P}'_1} f''(P) \leq \sum_{e \in E'_S} \hat{c}(e) = \hat{c}_S = F$ from Claim 6.3. From Corollary 6.6, $\sum_{P \in \mathcal{P}'_2} f''(P) \leq 2^{14} \cdot \gamma \cdot M' \cdot n^{\epsilon} \cdot \log^2 n \cdot \log(W_{\max}) = \eta$. Therefore:

$$\text{val}(f) \leq \sum_{P \in \mathcal{P}'_1} f''(P) + \sum_{P \in \mathcal{P}'_2} f''(P) \leq F + \eta.$$

□

6.3 Flow Deficit and Bad Batches of Vertices.

Recall that we have denoted by Q'_0 the set of all vertices $v \in U$ with $v^{\text{in}} \notin V(J)$, and that, from Claim 6.7 if (s, T) is a good pair, all shortcut operations so far are valid, and Event $\hat{\mathcal{E}}$ did not happen, then $|Q'_0 \cap (S \cup L)| \leq n^{1-8\epsilon}$. We now denote by Q_0 all remaining vertices of U, that is, vertices $v \in U$ with $v^{\text{in}} \in V(J)$. Our goal is to compute a partition (Q, Q') of U that has the following property: on the one hand, only a small number of vertices of Q' belong to $S \cup L$; on the other hand, in any maximum s^{out}-t' flow in $\tilde{H}'$, for most vertices $v \in Q$, the total flow on the parallel edges (v, t') is close to their capacity. We will ensure that $Q'_0 \subseteq Q'$, so it now remains to compute the partition of Q_0. A central notion that we use in in order to compute the partition is that of a *flow deficit*, which we define next.

Flow deficit. For every vertex $v \in Q_0$, we denote by $A(v)$ the set of all edges $(v^{\text{in}}, x) \in E(H)$ with $x \notin V(J)$, and by $\beta(v)$ their total capacity in H. Notice that every edge $(v^{\text{in}}, x) \in A(v)$ is represented

by a distinct edge (v^{in}, t') in $\tilde{H}'$; we do not distinguish between these edges, so sometimes we may equivalently view $A(v)$ as the set of all parallel edges (v^{in}, t') of $\tilde{H}'$. Given an s^{out}-t' flow f' in $\tilde{H}'$, for every vertex $v \in Q_0$, the *flow deficit of* v, denoted by $\Delta_{f'}(v)$, is $\beta(v) - \sum_{e \in A(v)} f'(e)$.

For convenience, for any subset $\hat{Q} \subseteq Q_0$ of vertices, we will denote by $\beta(\hat{Q}) = \sum_{v \in \hat{Q}} \beta(v)$, and by $A(\hat{Q}) = \bigcup_{v \in \hat{Q}} A(v)$.

Intuitively, the flow deficit measures the amount of the capacity of the edges of $A(v)$ that is unused by the flow. The following claim is key to the second step. Intuitively, the claim shows that, if we are given any maximum s^{out}-t' flow in $\tilde{H}'$, then the total deficit of the vertices of $Q_0 \cap S$ must be small. This property will be used in order to identify subsets $\hat{B} \subseteq Q_0$ of vertices that only contain relatively few vertices of S; we will refer to such subsets as *bad batches* of vertices.

Claim 6.8 *Assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let f' be any maximum s^{out}-t' flow in $\tilde{H}'$. Then the total flow deficit of the vertices of $Q_0 \cap S$ is $\sum_{v \in Q_0 \cap S} \Delta_{f'}(v) \leq 2\eta$.*

Proof: Assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let f' be any maximum s^{out}-t' flow in $\tilde{H}'$. From Claim 6.7 and Claim 6.3, the value of the flow is $F' \geq F = \hat{c}_S = \sum_{e \in E'_S} \hat{c}(e)$. Let $\mathcal{P}$ be a flow-path decomposition of f', and let $(\mathcal{P}', \mathcal{P}'')$ be a partition of $\mathcal{P}$, where set $\mathcal{P}''$ contains all problematic paths, and set $\mathcal{P}'$ contains all remaining paths. From Corollary 6.6, $\sum_{P \in \mathcal{P}''} f'(P) \leq 2^{14} n^{\epsilon} \cdot M' \cdot \gamma \cdot \log^2 n \cdot \log(W_{\max}) = \eta$. Therefore, $\sum_{P \in \mathcal{P}'} f'(P) \geq F' - \eta \geq \sum_{e \in E'_S} \hat{c}(e) - \eta$. Note that, from the definition of problematic paths, every path $P \in \mathcal{P}'$ contains an edge of E'_S or its copy; in the discussion below, we do not distinguish between edges of E'_S and their copies in $\tilde{H}'$. Therefore: $\sum_{e \in E'_S} f'(e) \geq \sum_{P \in \mathcal{P}'} f'(P) \geq \sum_{e \in E'_S} \hat{c}(e) - \eta$. Altogether, we get that:

$$\sum_{e \in E'_S} (\hat{c}(e) - f'(e)) \leq \eta.$$

Let $A' = \bigcup_{v \in Q_0 \cap S} A(v)$, and let $E''_S = A' \cap E'_S$. Since, for every edge $e \in E'_S$, $f'(e) \leq \hat{c}(e)$, we get that:

$$\sum_{e \in E''_S} (\hat{c}(e) - f'(e)) \leq \sum_{e \in E'_S} (\hat{c}(e) - f'(e)) \leq \eta.$$

Since $E''_S \subseteq A'$, we get that:

$$\sum_{e \in A'} (\hat{c}(e) - f'(e)) = \sum_{e \in A' \setminus E''_S} (\hat{c}(e) - f'(e)) + \sum_{e \in E''_S} (\hat{c}(e) - f'(e)) \leq \sum_{e \in A' \setminus E''_S} \hat{c}(e) + \eta. \tag{13}$$

Finally, we bound $\sum_{e \in A' \setminus E''_S} \hat{c}(e)$ in the following observation.

Observation 6.9 *Assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Then $\sum_{e \in A' \setminus E''_S} \hat{c}(e) \leq \eta$.*

Proof: Consider some vertex $v \in Q_0 \cap S$, and recall that we have denoted by $A(v)$ the set of all edges $(v^{\text{in}}, x) \in E(H)$ with $x \notin V(J)$. Consider now any edge $e = (v^{\text{in}}, x) \in A(v)$. From the construction of the split graph, $x \in V^{\text{out}}$ must hold, so we denote $x = u^{\text{out}}$ for some vertex $u \in V(G)$. If $u = v$, then

edge e lies in E^0_S, and if $u \in (R \cup S) \setminus \{v\}$, then $e \in E^1_S \cup E^2_S$. Therefore, the only way that $e \notin E'_S$ is if $u \in L$. We conclude that for every edge $e = (x, y)$ with $e \in A' \setminus E''_S$, it must be the case that $x \in V(J)$ and $y \in L^* \setminus V(J)$. From Corollary 5.7, $\sum_{e \in A' \setminus E''_S} \hat{c}(e) \leq 2^{14} \cdot n^{\epsilon} \cdot \gamma \cdot M' \cdot \log^2 n \cdot \log(W_{\max}) = \eta$. □

Altogether, we get that:

$$\sum_{v \in Q_0 \cap S} \Delta_{f'}(v) = \sum_{e \in A'} (\hat{c}(e) - f'(e)) \leq \sum_{e \in A' \setminus E''_S} \hat{c}(e) + \eta \leq 2\eta.$$

□

Recall that $M' = M_i = \frac{W'_{\max}}{2^i \cdot \gamma}$, and that the number of phases in the algorithm is bounded by $z = \left\lceil \log \left(\frac{8W'_{\max} \cdot n^2}{\gamma} \right) \right\rceil$. Therefore, $M' \geq M_z \geq \frac{W'_{\max}}{\gamma \cdot 2^z}$, and $\log(M') \geq -8 \log(W_{\max})$. We define a parameter j_0 as follows: if $M' \geq 1$, then $j_0 = 0$, and otherwise, $j_0 = \log(M')$. From our discussion:

$$-8 \log(W_{\max}) \leq j_0 \leq 0. \tag{14}$$

Notice that, since the capacity of every regular edge in G'' is $W_{\max}$, from Property Q'1, and since the flow f is M'-integral, the capacity of every edge in H is at least 2^{j_0}.

For an integer $j \geq j_0$, let $B_j = \left\{ v \in Q_0 \mid \beta(v) \geq 2^j \right\}$; for brevity, we will refer to the vertices of B_j as *j-interesting* vertices. We next bound the number of j-interesting vertices that may lie in S, for all such j. The following claim shows that for sufficiently large values of j, $|B_j \cap S|$ must be small, and so in particular such a set B_j of vertices may only contain few vertices of $S \cup L$.

Claim 6.10 *Assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Then for all $j \geq j_0$, $|B_j \cap S| \leq \frac{9n \cdot M'}{2^j}$.*

Proof: Assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Fix an integer $j \geq j_0$, and consider a maximum s^{out}-t' flow f' in graph $\tilde{H}'$. Let $\mathcal{P}$ be a flow-path decomposition of f'.

Let $A' = A(B_j \cap S) = \bigcup_{v \in B_j \cap S} A(v)$. Note that, in graph $\tilde{H}'$, every edge of A' connects a vertex v^{in}, for $v \in B_j \cap S$, to the vertex t'. Therefore, we can assume w.l.o.g. that every flow-path in $\mathcal{P}$ contains at most one edge of A'. From Claim 6.7 and Claim 6.3:

$$\text{val}(f') = F' \leq F + \eta \leq 8nM' + \eta.$$

Therefore:

$$\sum_{e \in A'} f'(e) \leq \text{val}(f') \leq 8nM' + \eta.$$

We can then lower-bound the total deficit of the vertices of $B_j \cap S$ with respect to f' as follows:

$$\begin{aligned}
\sum_{v \in B_j \cap S} \Delta_{f'}(v) &= \sum_{v \in B_j \cap S} \left(\beta(v) - \sum_{e \in A(v)} f'(e) \right) \\
&\geq |B_j \cap S| \cdot 2^j - \sum_{e \in A'} f'(e) \\
&\geq |B_j \cap S| \cdot 2^j - 8nM' - \eta.
\end{aligned}$$

On the other hand, from Claim 6.8, $\sum_{v\in B_j\cap S}\Delta_{f'}(v)\leq 2\eta$ must hold.

Therefore, we get that:

$$|B_j\cap S|\leq \frac{3\eta}{2^j}+\frac{8nM'}{2^j}\leq\frac{9nM'}{2^j},$$

since $\eta=2^{20}n^{\epsilon}\cdot M'\cdot\gamma\cdot\log^2 n\cdot\log(W_{\max})\leq nM'$, as $\gamma\leq 2n^{1-10\epsilon}$ from Inequality 7 and $n^{\epsilon}\geq 2^{14}\log^2 n\log(W_{\max})$ from Inequality 5. □

Let j^* be the smallest integer for which $2^{j^*}\geq 4n^{9\epsilon}\cdot M'$ holds (and notice that j^* may be negative). Clearly:

$$4n^{9\epsilon}\cdot M'\leq 2^{j^*}\leq 8n^{9\epsilon}\cdot M'. \tag{15}$$

Additionally, since $\gamma\geq n^{10\epsilon}$ from Inequality 8, we get that:

$$2^{j^*}\leq\frac{\gamma\cdot M'}{16}. \tag{16}$$

Let $Q_1'\subseteq Q_0$ be the collection of all vertices $v\in Q_0$ with $\beta(v)\geq 2^{j^*}$; equivalently, $Q_1'=B_{j^*}$. Then, from Claim 6.10, if (s,T) is a good pair, all shortcut operations so far have been valid, and Event $\hat{\mathcal{E}}$ did not happen, it must be the case that:

$$|Q_1'\cap(S\cup L)|\leq|Q_1'\cap S|+|Q_1'\cap L|\leq\frac{9nM'}{2^{j^*}}+n^{\epsilon}\leq 2n^{1-8\epsilon}. \tag{17}$$

We are now ready to define a central notion for the current step: the j-bad batches of vertices.

Definition 6.11 (j-bad batch of vertices.) *Let f' be any maximum s^{out}–t' flow in $\tilde{H}'$, and let $j_0\leq j<j^*$ be an integer. A subset $\hat{B}\subseteq Q_0\setminus Q_1'$ of vertices is a* j-bad batch *with respect to f' if the following hold:*

- *for every vertex $v\in\hat{B}$, $\Delta_{f'}(v)\geq 2^j$; and*
- $|\hat{B}|\geq\frac{n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^j}$.

In the next key claim we show that only a small fraction of vertices in a j-bad batch may lie in $S\cup L$.

Claim 6.12 *Assume that (s,T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let $\hat{B}\subseteq Q_0\setminus Q_1'$ be a j-bad batch of vertices with respect to some maximum s^{out}–t' flow f' in $\tilde{H}'$, for some $j_0\leq j<j^*$. Then $|\hat{B}\cap(S\cup L)|\leq\frac{|\hat{B}|}{n^{4\epsilon}}$.*

Proof: Assume that (s,T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let $\hat{B}\subseteq Q_0\setminus Q_1'$ be a j-bad batch of vertices with respect to some maximum s^{out}–t' flow f' in $\tilde{H}'$, for some $j_0\leq j<j^*$. Denote by $B'=\hat{B}\cap(S\cup L)$, and assume for contradiction that $|B'|>\frac{|\hat{B}|}{n^{4\epsilon}}$. Then:

$$|B'|>\frac{|\hat{B}|}{n^{4\epsilon}}\geq\frac{n^{1.5\epsilon}\cdot\gamma\cdot M'}{2^j}.$$

Recall that $\gamma \cdot M' \geq 2^{j^*} \geq 2^j$, from Inequality 16. Therefore, $|B' \cap L| \leq n^{\epsilon} \leq \frac{|B'|}{2}$ (we have used Inequality 3). But then we get that:

$$\sum_{v \in B' \cap S} \Delta_{f'}(v) \geq 2^j \cdot |B' \cap S| \geq \frac{n^{1.5\epsilon}\gamma \cdot M'}{2} > 2\eta,$$

contradicting Claim 6.8 (we have used the fact that $\eta = 2^{14} n^{\epsilon} \cdot M' \cdot \gamma \cdot \log^2 n \cdot \log(W_{\max})$, and that $n^{\epsilon/10} > 2^{20} \log^2 n \cdot \log(W_{\max})$ from Inequality 5) □

Let $Q_2' \subseteq Q_0$ contain all vertices $v \in Q_0$ with $v^{\text{out}} \in V(J)$. It is easy to verify that:

$$|Q_2'| \leq N = \frac{32n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma} \leq n^{1-5\epsilon}, \tag{18}$$

since $N = \frac{32n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}$, $\gamma \geq n^{8\epsilon}$ from Inequality 8, and $n^{\epsilon} \leq 2^{20} \log n \log(W_{\max})$ from Inequality 5.

6.4 The Algorithm

We are now ready to describe the algorithm for Step 2. Our algorithm works with graph $\tilde{H}$ – the graph that is obtained from $\tilde{H}'$ by unifying the sets of parallel edges that are incident to t'. Note that, for every vertex $v \in Q_0$, $\beta(v)$ is equal to the capacity of the edge (v, t') in graph $\tilde{H}$. We compute the set $Q_1' = B_{j^*}$ of vertices, and we initialize $Q = Q_0 \setminus (Q_1' \cup Q_2')$ and $Q' = Q_0' \cup Q_1' \cup Q_2'$. Notice that (Q, Q') is a partition of U.

Our algorithm performs a number of iterations. In every iteration, we identify some j-bad batches $\hat{B} \subseteq Q$ of vertices, for some $j_0 \leq j < j^*$, and move the vertices of each such batch $\hat{B}$ from Q to Q'. We ensure that all such batches of vertices are disjoint.

Specifically, in order to execute a single iteration, we set up an instance of maximum min-cost flow in graph $\tilde{H}$, with source s^{out} and destination t'. For every vertex v that currently lies in Q, we set the *cost* of the edge (v, t') to be 1, and the costs of all other edges of $\tilde{H}$ are set to 0. We then compute a maximum min-cost flow from s^{out} to t' in $\tilde{H}$, using the algorithm from Theorem 2.4, obtaining an integral flow f'. For every vertex $v \in Q$, we compute its deficit $\Delta_{f'}(v) = \beta(v) - f'(v, t')$. For all $j_0 \leq j < j^*$, we let $\hat{B}_j \subseteq Q$ be the set of all vertices $v \in Q$ with $2^j \leq \Delta_{f'}(v) < 2^{j+1}$. For every integer $j_0 \leq j < j^*$ for which $|\hat{B}_j| \geq \frac{n^{5.5\epsilon} \cdot \gamma \cdot M'}{2^j}$ holds, we move all vertices of $\hat{B}_j$ from Q to Q'; notice that, in this case, $\hat{B}_j$ is a j-bad batch with respect to f'. If, for any integer $j_0 \leq j < j^*$, $|\hat{B}_j| \geq \frac{n^{5.5\epsilon} \cdot \gamma \cdot M'}{2^j}$ holds, then we say that the current iteration is of *type 1*. Otherwise, for all $j_0 \leq j < j^*$, $|\hat{B}_j| < \frac{n^{5.5\epsilon} \cdot \gamma \cdot M'}{2^j}$ holds, and we say that the current iteration is of *type 2*. In this case, the current iteration becomes the last one. Notice that the time required to execute every iteration is bounded by $O\left(|E(\tilde{H})|^{1+o(1)} \cdot \log(W_{\max})\right)$. Since every edge of $\tilde{H}$ is either incident to t' or has an endpoint in Γ, we get that: $|E(\tilde{H})| \leq O(N \cdot n) \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right)$, since $N = \frac{32n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}$. Altogether, the running time of a single iteration is $O\left(\frac{n^{2+2\epsilon+o(1)} \cdot \log^3(W_{\max})}{\gamma}\right)$. We bound the number of iterations in the following claim.

Claim 6.13 *If (s, T) is a good pair, all shortcut operations so far have been valid, and the event $\hat{\mathcal{E}}$ did not happen, then the total number of type-1 iterations is at most $\frac{216n^{1-5.5\epsilon} \cdot \log n \cdot \log(W_{\max})}{\gamma}$.*

Proof: For every integer $j_0 \leq j < j^*$, let $B'_j = B_j \setminus B_{j+1}$, so B'_j contains all vertices $v \in Q_0$, with $2^j \leq \beta(v) < 2^{j+1}$. The following observation is key to proving the claim.

Observation 6.14 *Assume that (s,T) is a good pair, all shortcut operations so far have been valid, and the event $\hat{\mathcal{E}}$ did not happen. Then for every integer $j_0 \leq j < j^*$, after the first iteration is completed, $|B'_j \cap Q| \leq \frac{64nM'}{2^j}$ holds.*

Proof: Assume that (s,T) is a good pair, all shortcut operations so far have been valid, and the event $\hat{\mathcal{E}}$ did not happen. We fix an integer $j_0 \leq j < j^*$ and prove the claim for this integer. Let f' be the flow computed in the first iteration. We partition the vertices of $B'_j \cap Q$ into two subsets: set X containing all vertices v with $\Delta_{f'}(v) \geq 2^{j-1}$, and set X' containing all remaining vertices.

Observe first that $|X| \leq \frac{4n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^{j-1}}$ must hold. Indeed, assume otherwise, so $|X| > \frac{4n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^{j-1}}$. Since, for every vertex $v \in B'_j$, $\beta(v) < 2^{j+1}$, we get that $\Delta_{f'}(v) < 2^{j+1}$ must hold as well. Therefore, for every vertex $v \in X$, $2^{j-1} \leq \Delta_{f'}(v) < 2^{j+1}$ holds. We let $X_1 \subseteq X$ be the set of all vertices $v \in X$ with $2^{j-1} \leq \Delta_{f'}(v) < 2^j$, and we let X_2 be the set of all remaining vertices, so for each vertex $v \in X_2$, $2^j \leq \Delta_{f'}(v) < 2^{j+1}$ holds. If $|X_1| \geq \frac{|X|}{2} > \frac{2n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^{j-1}}$ holds, then, since $X_1 \subseteq \hat{B}_{j-1}$, the set $\hat{B}_{j-1}$ of vertices should have been moved from Q to Q', including all vertices of X_1. Otherwise, $|X_2| \geq \frac{|X|}{2} > \frac{4n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^j}$ holds, then, since $X_1 \subseteq \hat{B}_{j-1}$, the set $\hat{B}_{j-1}$ of vertices should have been moved from Q to Q', a contradiction. We conclude that $|X| \leq \frac{4n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^{j-1}}$.

Consider now the set X' of vertices. Observe that, for every vertex $v \in X'$, $\Delta_{f'}(v) < 2^{j-1}$, and, since $\beta(v) \geq 2^j$, we get that $f'(v,t') \geq 2^{j-1}$. Overall, we get that:

$$\mathrm{val}(f') \geq \sum_{v\in X'} f'(v,t') \geq |X'|\cdot 2^{j-1}.$$

On the other hand, from Claim 6.7 and Claim 6.3, if (s,T) is a good pair, all shortcut operations so far have been valid and Event $\hat{\mathcal{E}}$ did not happen:

$$\mathrm{val}(f') = F' \leq F + \eta \leq 8nM' + \eta \leq 16nM',$$

since $\eta = 2^{14}n^{\epsilon}\cdot M'\cdot\gamma\cdot\log^2 n\cdot\log(W_{\max})$, $\gamma \leq n^{1-2\epsilon}$ from Inequality 7, and $2^{14}\cdot\log^2 n\cdot\log(W_{\max}) \leq n^{\epsilon}$ from Inequality 5. Altogether we get that $|X'| \leq \frac{16nM'}{2^{j-1}}$, and:

$$|B'_j \cap Q| \leq |X| + |X'| \leq \frac{4n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^{j-1}} + \frac{16nM'}{2^{j-1}} \leq \frac{64nM'}{2^j},$$

since $\gamma \leq 2n^{1-10\epsilon}$ from Inequality 7. □

Since, for all $j_0 \leq j < j^*$, $B_j = \bigcup_{j'=j}^{j^*-1} B'_j$, we get that, after the first iteration, for all $j_0 \leq j < j^*$, $|B_j \cap Q| \leq \frac{128nM'}{2^j}$ holds. Consider now some iteration of the algorithm that is a type-1 iteration. Then there must be at least one integer $j_0 \leq j < j^*$, for which, in the current iteration, $|\hat{B}_j| \geq \frac{n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^j}$ held. We then say that this iteration is a *type-1(j) iteration* (if there are several such integers j, we choose one arbitrarily.) Recall that $\hat{B}_j$ contains all vertices $v \in Q$ with $2^j \leq \Delta_{f'}(v) < 2^{j+1}$, so in particular $\beta(v) \geq 2^j$ for each such vertex v, and therefore, $\hat{B}_j \subseteq B_j$ must hold. If some iteration r is a type-1(j) iteration, then the vertices of $\hat{B}_j$ are deleted from Q during that iteration. Since, after the first iteration, $|B_j \cap Q| \leq \frac{128nM'}{2^j}$, and since each type-1($j$) iteration moves at least $\frac{n^{5.5\epsilon}\cdot\gamma\cdot M'}{2^j}$ vertices of B_j from Q to Q', we get that the total number of type-1(j) iterations is bounded by:

$$\frac{128nM'}{2^j} \cdot \frac{2^j}{n^{5.5\epsilon} \cdot \gamma \cdot M'} \leq \frac{128n^{1-5.5\epsilon}}{\gamma}.$$

Summing this up over all $j_0 \leq j < j^*$, and recalling that $j^* \leq \log n \log(W_{\max})$ from Inequality 16, and $j_0 \geq -8\log(W_{\max})$, from Inequality 14, we get that the total number of iterations is bounded by $\frac{216n^{1-5.5\epsilon} \cdot \log n \cdot \log(W_{\max})}{\gamma}$. □

We execute the iterations of min-cost flow computations until either a type-2 iteration is encountered, or more than $\frac{256n^{1-5.5\epsilon} \cdot \log n \cdot \log(W_{\max})}{\gamma}$ type-1 iterations occur. In the latter case, we terminate the algorithm for the current phase with a "FAIL". The following observation is immediate.

Observation 6.15 *The algorithm for Step 2 may only terminate with a "FAIL" if either (s,T) is not a good pair, or at least one shortcut operation executed before the current phase or during Step 1 of the current phase is invalid, or Event $\hat{\mathcal{E}}$ has happened.*

Since the running time of every iteration is $O\left(\frac{n^{2+2\epsilon+o(1)} \cdot \log^3(W_{\max})}{\gamma}\right)$, we get that the total running time of Step 2 so far is:

$$\begin{aligned} O\left(\frac{n^{2+2\epsilon+o(1)} \cdot \log^3(W_{\max})}{\gamma}\right) \cdot O\left(\frac{n^{1-5.5\epsilon} \cdot \log n \cdot \log(W_{\max})}{\gamma}\right) \\ \leq O\left(\frac{n^{3-3.5\epsilon+o(1)} \cdot \log^4(W_{\max})}{\gamma^2}\right) \\ \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right), \end{aligned}$$

since $\gamma \geq n^{2/3+5\epsilon}$ from Inequality 7.

We now summarize the properties of the final partition (Q, Q') of U that the algorithm obtains. We start by showing that Q' may only contain a relatively small number of vertices from $S \cup L$.

Claim 6.16 *Assume that (s,T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Then $|Q' \cap (S \cup L)| \leq 2n^{1-4\epsilon}$.*

Proof: We assume that (s,T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen. Then from Claim 6.7 $|Q'_0 \cap (S \cup L)| \leq n^{1-8\epsilon}$. Additionally, from Inequality 17, $|Q'_1 \cap (S \cup L)| \leq 2n^{1-8\epsilon}$, and from Inequality 18, $|Q'_2| \leq n^{1-5\epsilon}$. Altogether:

$$|(Q'_0 \cup Q'_1 \cup Q'_2) \cap (S \cup L)| \leq 2n^{1-5\epsilon}.$$

For conveninence, denote $Q'_3 = Q' \setminus (Q'_0 \cup Q'_1 \cup Q'_2)$. From our algorithm, there is a partition $\mathcal{B}$ of Q'_3 that has the following property: for every set $\hat{B} \in \mathcal{B}$, there is an integer $j_0 \leq j < j^*$ and a maximum s^{out}-t' flow f' in $\tilde{H}'$, such that $\hat{B}$ is a j-bad batch for f'. From Claim 6.12, for each such batch $\hat{B} \in \mathcal{B}$, $|\hat{B} \cap (S \cup L)| \leq \frac{|\hat{B}|}{n^{4\epsilon}}$. Altogether, we get that:

$$|Q'_3 \cap (S \cup L)| = \sum_{\hat{B} \in \mathcal{B}} |\hat{B} \cap (S \cup L)| \leq \sum_{\hat{B} \in \mathcal{B}} \frac{|\hat{B}|}{n^{4\epsilon}} \leq n^{1-4\epsilon}.$$

Altogether:

$$|Q' \cap (S \cup L)| = |(Q'_0 \cup Q'_1 \cup Q'_2) \cap (S \cup L)| + |Q'_3 \cap (S \cup L)| \leq 2n^{1-5\epsilon} + n^{1-4\epsilon} \leq 2n^{1-4\epsilon}.$$

□

Let f^* be the min-cost maximum s^{out}-t' flow in $\tilde{H}$ computed in the last iteration, so the value of the flow must be F'. This flow naturally defines an s^{out}-t' flow in $\tilde{H}'$ of the same value, that we also denote by f^*. We also denote by $F^* = \sum_{e \in A(Q)} f^*(e)$ – the total flow that f^* sends on the edges of $A(Q) = \bigcup_{v \in Q} A(v)$. We summarize the central properties of the set Q of vertices in the following claim.

Claim 6.17 *$F^* \geq \beta(Q) - 4M' \cdot n^{5.5\epsilon} \cdot \gamma \cdot \log n \cdot \log(W_{\max})$. Moreover, if $\hat{f}$ is any maximum s^{out}-t' flow in $\tilde{H}'$, then $\sum_{e \in A(Q)} \hat{f}(e) \geq F^*$.*

Proof: The second assertion follows immediately from the definition of the min-cost flow: recall that the costs of all edges of $\tilde{H}'$ that represent the edges of $A(Q)$ are set to 1, while the costs of all other edges are 0. If there exists a maximum s^{out}-t' flow $\hat{f}$ in $\tilde{H}'$ with $\sum_{e \in A(Q)} \hat{f}(e) < F^*$, then the cost of this flow would be lower than that of f^*, a contradiction.

We now turn to prove the first assertion. As before, for all $j_0 \leq j < j^*$, we let $\hat{B}_j \subseteq Q$ be the set of all vertices $v \in Q$ with $2^j \leq \Delta_{f^*}(v) < 2^{j+1}$. Since the iteration where f^* was computed is of type 2, we get that, for all $j_0 \leq j < j^*$, $|\hat{B}_j| < \frac{n^{5.5\epsilon} \cdot \gamma \cdot M'}{2^j}$ holds. Therefore, for all $j_0 \leq j < j^*$:

$$\sum_{v \in \hat{B}_j} \left(\beta(v) - \sum_{e \in A(v)} f^*(e) \right) \leq 2^{j+1} \cdot \frac{n^{5.5\epsilon} \cdot \gamma \cdot M'}{2^j} \leq 2M' n^{5.5\epsilon} \gamma \tag{19}$$

Let $\hat{Q} = Q \setminus \left(\bigcup_{j=j_0}^{j^*-1} \hat{B}_j \right)$. Recall that, if $v \in Q$, then $\beta(v) < 2^{j^*}$ must hold (as otherwise v should have been added to Q'_1), and so $\Delta_{f^*}(v) < 2^{j^*}$ must hold. Since the flow f^* is M'-integral, and $j_0 = \log M'$, we get that, for every vertex $v \in \hat{Q}$, $\Delta_{f^*}(v) = 0$, and $\beta(v) = \sum_{e \in A(v)} f^*(e)$. By combining this with Inequality 19 for all $j_0 \leq j < j^*$, and recalling that $j^* \leq \log n \cdot \log(W_{\max})$, while $j_0 \geq -8 \log(W_{\max})$ from Inequalities 14 and 16, we get that:

$$\sum_{v \in Q} \left(\beta(v) - \sum_{e \in A(v)} f^*(e) \right) \leq 2M' \cdot n^{5.5\epsilon} \cdot \gamma \cdot (j^* - j_0) \leq 4M' \cdot n^{5.5\epsilon} \cdot \gamma \cdot \log n \cdot \log(W_{\max}),$$

and so:

$$F^* = \sum_{v \in Q} \sum_{e \in A(v)} f^*(e) \geq \beta(Q) - 4M' \cdot n^{5.5\epsilon} \cdot \gamma \cdot \log n \cdot \log(W_{\max}).$$

□

6.5 The Final Cut

We say that Case 1 happens if $|Q| > n^{1-4\epsilon}$, and otherwise we say that Case 2 happens. If Case 1 happens, then we need to perform one additional step that we describe below, after we prove the following simple observation.

Observation 6.18 *If Case 1 happened, then* $F^* \geq n^{1-4\epsilon} \cdot M'/2$.

Proof: Recall that for every vertex $v \in Q$, $v^{\text{in}} \in V(J)$ and $v^{\text{out}} \notin V(J)$ holds (by the definition of the sets Q'_0 and Q'_2 of vertices), so the special edge $(v^{\text{in}}, v^{\text{out}})$ belongs to $A(v)$. From Property Q'1, the capacity of this edge in H is at least M', and so $\beta(v) \geq M'$ must hold. Since we assumed that Case 1 happened, $|Q| \geq n^{1-4\epsilon}$, so $\beta(Q) \geq n^{1-4\epsilon} \cdot M'$. From Claim 6.17:

$$F^* \geq \beta(Q) - 4M'n^{5.5\epsilon} \cdot \gamma \cdot \log n \log(W_{\max}) \geq \frac{n^{1-4\epsilon} \cdot M'}{2},$$

since $n^{\epsilon/10} > 16 \log n \cdot \log(W_{\max})$ from Inequality 5 and $\gamma \leq 2n^{1-10\epsilon}$ from Inequality 7. □

Assume that Case 1 happened. We start by providing an intuitive motivation and explanation of the additional step required in this case. For the sake of this informal exposition, we assume that (s, T) is a good pair, that all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen.

In Step 3, we will sparsify the graph H, to compute a graph $H' \subseteq H$ (after possibly performing some shortcut operations). The goal of that step is to ensure that, on the one hand, $|E(H')|$ is quite small, while, on the other hand, the value of the maximum s^{out}-t flow in H' is close to F – the value of the maximum s^{out}-t flow in H. One of the main ideas in sparsifying the graph H is to use the fact that set Q' contains only a small number of vertices of $S \cup L$. Therefore, if a vertex $v \in V(G)$ has many neighbors in Q', then $v \in S \cup R$ must hold. We can then safely add a shortcut edge (v^{out}, t'), and delete all other edges leaving v^{out} from H. Additionally, in the sparsification step, we rely on the strong properties of the graph J that are summarized in Observation 5.6, and that ensure that the total number of edges of H with both endpoints in J, and of edges (x, y) with $x \in J$ and $y \notin J$, is relatively small. However, one major issue still remains, and it involves the set of regular edges $E^* = \left\{(u^{\text{out}}, v^{\text{in}}) \mid u^{\text{out}} \notin J, v^{\text{in}} \in J\right\}$. If $|Q|$ remains sufficiently large, then the number of such edges may be large, and it is not immediately clear how to sparsify them.

For further intuition, consider any maximum s^{out}-t flow f in H, and let $\mathcal{P}$ be a flow-path decomposition of f. For every path $P \in \mathcal{P}$, let $P' \subseteq P$ be the subpath of P starting from s^{out} and terminating at the first vertex of P that lies outside of $V(J)$. Let P'' be the subpath of P starting from the last vertex of P' and terminating at t. We say that path P is *meandering* with respect to $V(J)$, if P'' contains some vertex of $V(J)$. Notice that all paths in $\mathcal{P}$ that use the edges of E^* must be meandering. We also define a flow $\hat{f}$ in H, where for every path $P \in \mathcal{P}$, we send $f(P)$ flow units via the path P' (so in other words, we only send flow via the prefixes of the paths in $\mathcal{P}$).

Let $A' = A(Q)$, and let A'' contain all other edges $e = (x, y) \notin A'$ with $x \in V(J)$ and $y \notin V(J)$. Notice that our algorithm ensures that, in any maximum s^{out}-t' flow in $\tilde{H}'$, the total flow on the edges of A' is close to their total capacity. Assume for now that we could achieve the same property for the edges of A''. Consider now a maximum s^{out}-t flow f in H, and let the set $\mathcal{P}$ of paths and the flow $\hat{f}$ be defined as before. Since the value F of the flow f is quite close to the value F' of the maximum s^{out}-t' flow in $\tilde{H}$, the edges of $A' \cup A''$ must be almost saturated by the flow $\hat{f}$. If $P \in \mathcal{P}$ is a meandering path, then it needs to use at least two edges of $A' \cup A''$. It would then follow that the total flow on the meandering paths must be low, and so simply deleting the edges of E^* from the graph H would not significantly decrease the value of the maximum s^{out}-t flow in H.

Unfortunately, our algorithm only guarantees that, in any maximum s^{out}-t' flow f' in $\tilde{H}'$, the total flow on the edges of A' is close to the total capacity of these edges, but the flow on the edges of A'' may be much lower than their total capacity. This may potentially leave a lot of capacity that the meandering paths may exploit, and so we cannot claim directly that the total flow on the meandering paths is low. Therefore, deleting the edges of E^* from H may significantly reduce the value of the

maximum s^{out}-t flow in H. In fact it is not difficult to show that, in any maximum s^{out}-t flow in H, there is only a small subset $\hat{Q} \subseteq Q$ of vertices, such that the edges of E^* incident to the vertices of $\left\{v^{\text{in}} \mid v \in \hat{Q}\right\}$ carry any significant amount of flow (this follows from the properties of the min-cost flow). So only a relatively small number of edges of E^* are essential for approximately preserving the maximum s^{out}-t flow in H. But unfortunately it is not clear how to identify such edges.

In order to overcome this difficulty, we compute a minimum cut in graph $\tilde{H}$, separating s^{out} from the edges of A''; denote this cut by $(\hat{X}, \hat{Y})$, with $s^{\text{out}} \in \hat{X}$. We denote by $Q'_3 \subseteq Q$ the set of vertices $v \in Q$ with $v^{\text{in}} \in \hat{Y}$, and we show that $|Q'_3|$ is small. The vertices of Q'_3 are then deleted from Q and added to Q'. If we now consider the graph H^*, that is obtained from H by unifying all vertices of $V(H) \setminus \hat{X}$ into a vertex t'', then, as we show later, this graph now has the desired properties: the value of the maximum s^{out}–t'' flow in H^* is close to F, and in any such flow, all edges (x, y) with $x \in \hat{X}$ and $y \notin \hat{X}$ are close to being saturated. This allows us to delete from H all edges $(u^{\text{out}}, v^{\text{in}})$ with $u^{\text{out}} \notin \hat{X}$ and $v^{\text{in}} \in \hat{X}$, in order to sparsify the graph H, without significantly decreasing the value of the maximum s^{out}-t flow in it. We now describe the algorithm for the final cut more formally. This algorithm is only executed if Case 1 happens, that is, $|Q| > n^{1-4\epsilon}$ holds after the last iteration of the min-cost flow calculations.

Recall that we denoted by $A' = A(Q) = \bigcup_{v \in Q} A(v)$, and by A'' all edges $(x, y) \in E(H) \setminus A'$ with $x \in V(J)$ and $y \notin V(J)$. We let $\tilde{H}''$ be the graph that is very similar to $\tilde{H}'$, except for the following changes. We delete the sink t' and instead add two new sinks t_1 and t_2 to $\tilde{H}''$. For every edge $e = (x, y) \in A'$, we add the edge (x, t_1) of capacity $\hat{c}(e)$ representing e to $\tilde{H}''$, and for every edge $e' = (x', y') \in A''$, we add the edge (x', t_2) of capacity $\hat{c}(e')$ representing e' to $\tilde{H}''$.

We then compute a minimum s^{out}-t_2 cut $(\hat{X}, \hat{Y})$ (with respect to edge capacities) in $\tilde{H}''$, where $s^{\text{out}} \in \hat{X}$ and $t_2 \in \hat{Y}$. In order to compute the cut, we unify all parallel edges, obtaining a graph with $O(|E(\tilde{H})|)$ edges, and then apply the algorithm from Corollary 2.5 to the resulting graph. Note that the running time for this step is bounded by:

$$O(|E(\tilde{H}'')|) + O\left(|E(\tilde{H}|)^{1+o(1)} \cdot \log W_{\max}\right) \leq O(|E(\tilde{H}')|) + O\left(|E(\tilde{H}|)^{1+o(1)} \cdot \log W_{\max}\right).$$

It is easy to see that this step does not increase the asymptotic running time of Step 2, which remains bounded by $O\left(n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right)$.

Let $Q'_3 \subseteq Q$ be the set of all vertices $v \in Q$ with $v^{\text{in}} \in \hat{Y}$. Additionally, let $\hat{A}$ be the set of all edges $e = (x, y) \in E(H)$, with $x \in \hat{X}$ and $y \notin \hat{X}$. In the following claim, we summarize the main properties of the cut $(\hat{X}, \hat{Y})$.

Claim 6.19 *The total capacity $\sum_{e \in \hat{A}} \hat{c}(e) \leq F' + M' \cdot n^{1-4.4\epsilon}$. Additionally, $t_1 \in \hat{X}$, and $|Q'_3| \leq n^{1-4.4\epsilon}$.*

Proof: The following claim is central to the proof of Claim 6.19.

Claim 6.20 *The value of the maximum s^{out}-t_2 flow in $\tilde{H}''$ is equal to $F' - F^*$.*

Proof: Recall that we have computed an s^{out}-t' flow f^* in $\tilde{H}'$ of value F', where $\sum_{e \in A'} f^*(e) = F^*$, and so $\sum_{e \in A''} f^*(e) = F' - F^*$. This flow naturally defines an s^{out}-t_2 flow in $\tilde{H}''$ of value $F' - F^*$. Therefore, the value of the maximum s^{out}-t_2 flow in $\tilde{H}''$ is at least $F' - F^*$. It now remains to prove that it is at most $F' - F^*$.

Assume otherwise, that is, that the value of the maximum s^{out}-t_2 flow in $\tilde{H}''$ is greater than $F' - F^*$. Let $r = \min\{1, M'\}$. Since all edge capacities in G'' are integral, and the flow f is M'-integral, the value of the flow must be at least $F' - F^* + r$.

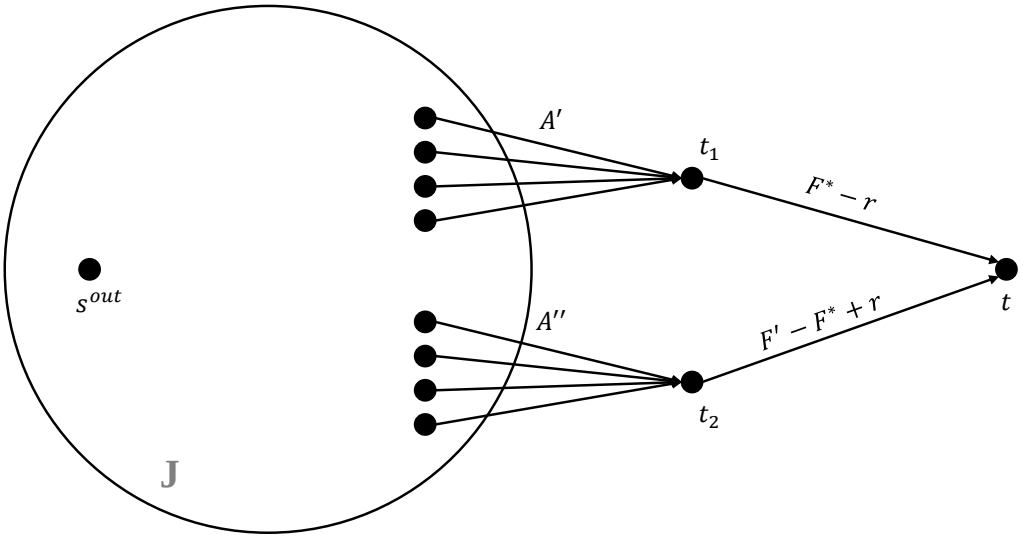


Figure 4: A schematic view of graph Z.

Let Z be the graph obtained from $\tilde{H}''$ by adding a single destination vertex t, and adding an edge (t_1, t) of capacity $F^* - r$, and an edge (t_2, t) of capacity $F' - F^* + r$ (see Figure 4). We use the following observation.

Observation 6.21 *If the value of the maximum s^{out}-t_2 flow in $\tilde{H}''$ is at least $F' - F^* + r$, then there is an s^{out}-t flow in Z of value F'.*

The proof of Claim 6.20 follows immediately from Observation 6.21. Indeed, assume for contradiction that the value of the maximum s^{out}-t_2 flow in $\tilde{H}''$ is greater than $F' - F^*$, so it is at least $F' - F^* + r$. Then, from Observation 6.21, there is an s^{out}-t flow in Z of value F'. This flow naturally defines an s^{out}-t' flow $\hat{f}$ in $\tilde{H}'$ of value F', in which $\sum_{e \in A'} \hat{f}(e) < F^*$, contradicting Claim 6.17. In order to complete the proof of Claim 6.20, it is now enough to prove Observation 6.21, that we do next.

Proof of Observation 6.21. For convenience, and in order to implify the notation, in this proof we denote s^{out} by s, and we denote the capacities of the edges $e \in E(Z)$ by $c(e)$. The proof provided here follows standard arguments, and it is very similar to the proof of Lemma 4.5 from [Chu16], which, in turn, relies on arguments suggested by Paul Seymour.

Assume for contradiction that the value of the maximum s-t flow in Z is less than F'. Then from the Max-Flow / Min-Cut Theorem, there is an s-t cut (X, Y) in Z with $\sum_{e \in E_Z(X,Y)} c(e) < F'$. Since the total capacity of the edges (t_1, t) and (t_2, t) is F', it must be the case that either $t_1 \in Y$, or $t_2 \in Y$ (or both).

Assume first that $t_1 \in X$ and $t_2 \in Y$. Then edge (t_1, t) contributes its capacity $F^* - r$ to the cut. Let E' denote the set of all edges $E_Z(X, Y)$, excluding the edge (t_1, t). Then $\sum_{e \in E'} c(e) = \sum_{e \in E_Z(X,Y)} c(e) - (F^* - r) < F' - F^* + r$. However, from our assumption, there exists an s-t_2 flow $\hat{f}$ in Z, whose value is at least $F' - F^* + r$. If we consider a flow-path decomposition $\mathcal{Q}$ of this flow, it is immediate to verify that every flow-path in the decomposition needs to use an edge of E', a contradiction.

Assume next that $t_2 \in X$ and $t_1 \in Y$. Then edge (t_2, t) contributes its capacity $F' - F^* + r$ to the cut. Let E'' denote the set of all edges $E_Z(X, Y)$, excluding the edge (t_2, t). Then $\sum_{e \in E''} c(e) = \sum_{e \in E_Z(X,Y)} c(e) - (F' - F^* + 1) < F^* - r$. Recall however that we have computed an s^{out}-t' flow in $\tilde{H}'$

of value F', where the total flow on the edges of A' is F^*. This flow can be naturally used to define an s-t_1 flow in Z of value F^*. If we consider a flow-path decomposition $\mathcal{Q}'$ of this flow, then every flow-path has to use an edge of E'', a contradiction.

Finally, assume that $t_1, t_2 \in Y$. Denote $E''' = E_Z(X,Y)$. Recall that graph $\tilde{H}'$ contains an s^{out}-t' flow f^* of value F'. This flow naturally defines a flow $\tilde{f}$ in Z of value F' from s to t_1 and t_2. In other words, if we consider a flow-path decomposition $\mathcal{Q}''$ of this flow, then every path originates at s and terminates at either t_1 or t_2. It is easy to verify that every flow-path in $\mathcal{Q}''$ must contain an edge of E'''. Since the value of the flow is F', while the total capacity of the edges in E''' is less than F', this is a contradiction. □

This completes the proof of Claim 6.20. □

We are now ready to prove that $t_1 \in \hat{X}$. Indeed, assume for contradiction that $t_1 \notin \hat{X}$. In this case, $\hat{A} = E_{\tilde{H}''}(\hat{X}, \hat{Y})$ holds; recall all edges of $\hat{A}$ lie in $\tilde{H}'$. Moreover, the total capacity of the edges of $\hat{A}$ in $\tilde{H}'$ is $F' - F^*$ by Claim 6.20, and from the Max-Flow / Min-Cut theorem. Additionally, from Observation 6.18, $F^* > 0$, and so $\sum_{e \in \hat{A}} \hat{c}(e) = F' - F^* < F'$. Notice however that the set $\hat{A}$ of edges separates s^{out} from t' in graph $\tilde{H}'$, contradicting the fact that there is an s^{out}-t' flow in $\tilde{H}'$ of value F'. We conclude that $t_1 \in \hat{X}$ must hold.

Let J' be the subgraph of $\tilde{H}''$ induced by the vertices of $\hat{X}$. In order to bound the cardinality of Q'_3, we need the following simple observation.

Observation 6.22 *There is an s^{out}–t_1 flow in J' of value at least F^*.*

Proof: For convenience, in this proof, we denote the capacities of the edges $e \in \tilde{H}''$ by $c(e)$. Assume otherwise. From the Max-Flow / Min-Cut theorem, there is a collection E_1 of edges in graph $\tilde{H}''$ with $\sum_{e \in E_1} c(e) < F^*$, such that in $J' \setminus E_1$ there is no path connecting s^{out} to t_1. Let $E_2 = E_{\tilde{H}''}(\hat{X}, \hat{Y})$, and recall that, from Claim 6.20 and the Max-Flow / Min-Cut theorem, $\sum_{e \in E_2} c(e) = F' - F^*$. Clearly, $\tilde{H}'' \setminus (E_1 \cup E_2)$ contains no paths connecting s to either t_1 or t_2, and, from our discussion, $\sum_{e \in E_1 \cup E_2} c(e) < F^* + F' - F^* = F'$. However, there is an s^{out}-t' flow f^* in $\tilde{H}'$ of value F', and this flow naturally defines a flow in $\tilde{H}''$ of value F' from s^{out} to t_1 and t_2. In other words, if $\mathcal{Q}$ is a flow-path decomposition of this flow, then every path in $\mathcal{Q}$ originates at s^{out} and terminates at t_1 or at t_2. Clearly, every path in $\mathcal{Q}$ has to contain an edge of $E_1 \cup E_2$. Since the total flow on the paths in $\mathcal{Q}$ is F', while the total capacity of the edges in $E_1 \cup E_2$ is less than F', this is a contradiction. □

Consider an s^{out}-t_1 flow $\hat{f}$ in J' of value at least F^*, and let $\mathcal{Q}$ be a flow-path decomposition of this flow. Note that, for every path $P \in \mathcal{Q}$, its penultimate vertex must belong to set $\left\{v^{\text{in}} \mid v \in Q \setminus Q'_3\right\}$, and so its last edge must lie in set $\bigcup_{v \in Q \setminus Q'_3} A(v)$. Therefore, $\beta(Q \setminus Q'_3) \geq F^*$ must hold. From Claim 6.17, $\beta(Q) \leq F^* + 4M' \cdot n^{5.5\epsilon} \cdot \gamma \cdot \log n \cdot \log(W_{\max})$. Therefore, we get that $\beta(Q'_3) = \beta(Q) - \beta(Q \setminus Q'_3) \leq 4M' \cdot n^{5.5\epsilon} \cdot \gamma \cdot \log n \cdot \log(W_{\max})$. Recall that, for every vertex $v \in Q$, $v^{\text{out}} \notin V(J)$, and so the special edge $(v^{\text{in}}, v^{\text{out}})$, whose capacity in H is at least M' (from Property Q'1), belongs to $A(v)$. Therefore, for every vertex $v \in Q$, $\beta(v) \geq M'$. We conclude that:

$$|Q'_3| \leq \frac{\beta(Q'_3)}{M'} \leq 4n^{5.5\epsilon} \cdot \gamma \cdot \log n \cdot \log(W_{\max}) \leq n^{1-4.4\epsilon},$$

since $n^{\epsilon/10} \geq 2^{20} \cdot \log n \cdot \log(W_{\max})$ from Inequality 5 and $\gamma \leq 2n^{1-10\epsilon}$ from Inequality 7.

Lastly, recall that we denoted by $\hat{A}$ all edges (x,y) of H with $x \in \hat{X}$ and $y \notin \hat{X}$. Since $t_2 \notin \hat{X}$ and $t_1 \in \hat{X}$, the set $\hat{A}$ of edges contains all edges of $E_{\tilde{H}''}(\hat{X}, \hat{Y})$ (where we do not distinguish between the edges of H and their copies in $\tilde{H}''$), and all edges in $\bigcup_{v \in Q \setminus Q'_3} A(v)$. By combining the Max-Flow /

Min-Cut theorem with Claim 6.20, we get that the total capacity of the edges in $E_{\tilde{H}''}(\hat{X},\hat{Y})$ is equal to $F'-F^*$. The total capacity of the edges in $\bigcup_{v\in Q\setminus Q'_3}A(v)$ is:

$$\beta(Q\setminus Q'_3)\leq\beta(Q)\leq F^*+4M'\cdot n^{5.5\epsilon}\cdot\gamma\cdot\log n\cdot\log(W_{\max})\leq F^*+M'\cdot n^{5.6\epsilon}\cdot\gamma\leq F^*+M'\cdot n^{1-4.4\epsilon}.$$

(we used Claim 6.17 and the fact that $n^{\epsilon/10}\geq 2^{20}\cdot\log n\cdot\log(W_{\max})$ from Inequality 5 and that $\gamma\leq 2n^{1-10\epsilon}$ from Inequality 7).

Altogether, we get that the total capacity of the edges of $\hat{A}$ in H is at most $F'+M'\cdot n^{1-4.4\epsilon}$. □

If Case 1 happens, then we compute the cut $(\hat{X},\hat{Y})$ in $\tilde{H}''$ as described above, and then move the vertices of Q'_3 from Q to Q'. Since, from Claim 6.19, $|Q'_3|\leq n^{1-4.4\epsilon}$, from Claim 6.16, if (s,T) is a good pair, all shortcut operations so far have been valid, and Event $\hat{\mathcal{E}}$ did not happen, then $|Q'\cap(S\cup L)|\leq 3n^{1-4\epsilon}$ holds. We also let $X^*=\hat{X}\cap V(H)$. Then we are guaranteed that $X^*\subseteq V(J)$, and $\sum_{e\in E_H(X^*,V(H)\setminus X^*)}\hat{c}(e)\leq F'+M'\cdot n^{1-4.4\epsilon}$. Recall that, from Claim 6.7, if (s,T) is a good pair, all shortcut operations so far have been valid, and Event $\hat{\mathcal{E}}$ did not happen, then:

$$F'\leq F+\eta=F+2^{14}n^{\epsilon}\cdot M'\cdot\gamma\cdot\log^2 n\cdot\log(W_{\max})\leq F+n^{1-8\epsilon}\cdot M'.$$

We now summarize the outcome of Step 2 in the following observation, whose proof follows immediately from our discussion.

Observation 6.23 *Let (Q,Q') be the partition of U computed in Step 2. Then the following hold:*

- *If (s,T) is a good pair, all shortcut operations so far have been valid, and Event $\hat{\mathcal{E}}$ did not happen, then $|Q'\cap(S\cup L)|\leq 3n^{1-4\epsilon}$; and*
- *If $|Q|\geq n^{1-4\epsilon}$, then the algorithm computed a subset $X^*\subseteq V(J)$ of vertices, such that, for every vertex $v\in Q$, $v^{\text{in}}\in X^*$ and $v^{\text{out}}\notin X^*$ holds. Additionally, if (s,T) is a good pair, all shortcut operations so far have been valid, and Event $\hat{\mathcal{E}}$ did not happen, then $\sum_{e\in E_H(X^*,V(H)\setminus X^*)}\hat{c}(e)\leq F+2n^{1-4.4\epsilon}\cdot M'$.*

7 Step 3: Sparsification

The goal of this step is to compute a subgraph $H'\subseteq H$, that is relatively sparse, so that the value of the maximum s^{out}-t flow in H' is close to that in H. We will then compute the maximum s^{out}-t flow in H', and augment f with this flow, obtaining the desired flow f_i that will serve as the output of the current phase.

We start by defining a partition of the edges of $E(H)$ into six subsets $E_1,\ldots,E_6$, and analyzing each subset separately. For some of these subsets we also describe the intuition for its sparsification. After that we formally define the sparsified graph H', and provide an algorithm for constructing H' efficiently. We also formally prove that, with high probability, the value of the maximum s^{out}-t flow in H' is sufficiently high. Lastly, we provide an algorithm for computing the desired output of the current phase.

7.1 Partition of $E(H)$

We start by defining a partition of $E(H)$ into six subsets $E_1,\ldots,E_6$, and by providing intuition for sparsifying some of these subsets. It may be convenient for now to think about constructing the graph

H' by starting with $H' = H$ and then gradually modifying it via the steps described below, though the eventual efficient algorithm for constructing H' will be different. We now define and analyze each of the sets in the partition in turn.

Set E_1: Recall that, for every vertex $v \in V(G) \setminus U$, $w(v) < M'$ must hold. We let E_1 be the set of all edges of H that are incident to the vertices of $\left\{v^{\text{in}}, v^{\text{out}} \mid v \in V(G) \setminus U\right\}$. Note that, since the current flow f is M'-integral, for every vertex $v \in V(G) \setminus U$, the flow on the special edge $(v^{\text{in}}, v^{\text{out}})$ must be 0. It is then easy to verify that the flow on every regular edge of G'' that enters v^{in} or leaves v^{out} is also 0. Therefore, every regular edge of E_1 must be of the form $(x^{\text{out}}, y^{\text{in}})$, where either $x \in V(G) \setminus U$, or $y \in V(G) \setminus U$, and this edge must also be present in G''. We will delete the edges of E_1 from the graph H'. It is easy to verify (and we do so formally later) that this may only decrease the value of the maximum s^{out}-t flow in H' by at most $M' \cdot n$.

Set E_2: Set E_2 contains all special edges of H (both the forward copies $(v^{\text{in}}, v^{\text{out}})$, and their backward counterparts $(v^{\text{out}}, v^{\text{in}})$), excluding those that lie in E_1. Additionally, E_2 contains all edges of H that are incident to t. Clearly, $|E_2| \leq 3n$. All edges of E_2 will be included in H'.

Set E_3: Set E_3 contains all regular edges (x, y) of H that do not lie in E_1, such that either (i) $x \in V(J)$; or (ii) at least one of the vertices x, y lies in $\Gamma = V(J) \cap V^{\text{out}}$ (note that both conditions may hold simultaneously). Consider any edge $e = (x, y) \in E_3$. Recall that $|\Gamma| \leq N = \frac{32n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}$. Therefore, the number of edges of E_3 that are incident to the vertices of Γ is bounded by $\frac{64n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma} \leq O(n^{2-4\epsilon})$, since $64 \log^2(W_{\max}) \leq n^{\epsilon}$ from Inequality 5, and $\gamma \geq n^{7\epsilon}$ from Inequality 8. Note that every edge with both endpoints in J must be incident to a vertex of Γ. It is then easy to verify that, if e is an edge of E_3 that is not incident to a vertex of Γ, then it must be the case that $e = (v^{\text{in}}, u^{\text{out}})$, where $v^{\text{in}} \in V^{\text{in}} \cap V(J)$ and $u^{\text{out}} \in V^{\text{out}} \setminus V(J)$. From Claim 5.5, each such edge must lie in set E^R, and, from Property P4 of Observation 5.6, $|E^R| \leq O\left(n \cdot \gamma \cdot \log n \cdot \log(W_{\max})\right) \leq O(n^{2-4\epsilon})$, since $\gamma \leq n^{1-5\epsilon}$ from Inequality 7, and $\log n \cdot \log(W_{\max}) \leq n^{\epsilon}$. Altogether, we get that $|E_3| \leq O(n^{2-4\epsilon})$. All edges of E_3 will be included in H'.

Set E_4: Set E_4 contains all regular edges (x, y) that do not lie in $E_1 \cup E_3$, such that $x \in V^{\text{out}}$ and $y \in \left\{v^{\text{in}} \mid v \in Q'\right\}$. Recall that, from Observation 6.23, if (s, T) is a good pair, all shortcut operations so far have been valid, and Event $\hat{\mathcal{E}}$ did not happen, then $|Q' \cap (S \cup L)| \leq 3n^{1-4\epsilon}$. Therefore, if $u^{\text{out}} \in V^{\text{out}}$ is a vertex that is incident to more than $3n^{1-4\epsilon}$ edges of E_4, then it must be the case that u has at least one neighbor in G that lies in R, and so $u \in R \cup S$ must hold. For each such vertex u^{out}, we can then add a shortcut edge (u^{out}, t) to G'', to H, and to H', and delete all other edges leaving u^{out} from H'. In order to execute this step efficiently, we will employ the δ-subgraph oracle with the set $Z = Q'$ of vertices; recall that the parameter δ was defined so that $n^{\delta} = \frac{n^{4\epsilon}}{4000 \log n}$ (see Equation 9). Recall that the oracle must return a partition (Y^h, Y^{ℓ}) of $V(G)$, and a collection E' of at most $n^{2-\delta} \cdot \log^2(W_{\max}) = n^{2-4\epsilon} \cdot 4000 \log n \cdot \log^2 W_{\max}$ edges of G, such that $E' = E_G(Y^{\ell}, Z)$ holds. If the oracle does not err, then every vertex of Y^h has at least $\frac{n^{1-\delta}}{1000 \log n} \geq 4n^{1-4\epsilon}$ neighbors in Q'. Therefore, if (s, T) is a good pair, all shortcut operations so far have been valid, Event $\hat{\mathcal{E}}$ did not happen and the oracle did not err, then every vertex $v \in Y^h$ must lie in $S \cup R$. For each such vertex $v \in Y^h$, we will add the shortcut edge (v^{out}, t) to G'', H, and H', and delete all edges of E_4 that leave the vertex v^{out} from H'. From the above discussion, the number of edges of E_4 that remain in H' is then bounded by $O(n^{2-4\epsilon} \cdot \log n \cdot \log^2(W_{\max}))$.

Set E_5. Set E_5 contains all regular edges $(x,y) \in E(H)$ that do not lie in $E_1 \cup E_3 \cup E_4$, such that $y = v^{\text{in}}$ for some vertex $v \in Q$, and $x \in V^{\text{out}} \setminus V(J)$. If $|Q| \leq n^{1-4\epsilon}$, then $|E_5| \leq |Q| \cdot n \leq n^{2-4\epsilon}$, and we keep all edges of E_5 in graph H'. Otherwise, all edges of E_5 will be deleted from H'. We will show that, if (s,T) is a good pair, all shortcut operations so far have been valid, and Event $\hat{\mathcal{E}}$ did not happen, then the deletion of the edges of E_5 from H' may only reduce the value of the maximum s^{out}-t flow in H' slightly. Intuitively, if $\hat{f}$ is the maximum s^{out}-t flow in H' before the edges of E_5 are deleted from it, and $\mathcal{P}$ is its flow-path decomosition, then every flow-path $P \in \mathcal{P}$ that contains an edge of E_5 must use at least two edges of $E_H(X^*, Y^*)$, where $Y^* = V(H) \setminus X^*$. However, from Observation 6.23, the total capacity of the edges in $E_H(X^*, Y^*)$ is close to F – the value of the maximum s^{out}-t flow in H. It then follows that the paths of $\mathcal{P}$ that contain edges of E_5 cannot carry large amount of flow in $\hat{f}$.

Set E_6. This set contains all remaining edges of H. The following observation provides a characterization of the edges of E_6.

Observation 7.1 *Let $e = (x,y)$ be an edge of E_6. Then e is a regular edge, with $x \in V^{\text{in}} \setminus V(J)$ and $y \in V^{\text{out}} \setminus V(J)$.*

Proof: Consider any edge $e = (x,y) \in E_6$. From the definition of set E_2, e must be a regular edge, and from the definition of set E_3, $x \notin V(J)$ must hold.

Assume first that $x = v^{\text{out}}$ and $y = u^{\text{in}}$ for some vertices $v^{\text{out}} \in V^{\text{out}}$ and $u^{\text{in}} \in V^{\text{in}}$. From the definition of set E_5, and since $x \notin V(J)$, we get that $u \notin Q$. From the definition of set E_1, and since (Q, Q') is a partition of U, we get that $u \in Q'$. But then the edge must lie in E_4.

We conclude that it must be the case that $x \in V^{\text{in}}$ and $y \in V^{\text{out}}$; denote $x = v^{\text{in}}, y = u^{\text{out}}$ for some $v, u \in V(G)$. From the definition of set E_3, we get that $v^{\text{in}} \notin J$ and $u^{\text{out}} \notin \Gamma$. We conclude that neither of the vertices $v^{\text{in}}, u^{\text{out}}$ may lie in J. □

Consider now any vertex $a^{\text{in}} \in V^{\text{in}} \setminus V(J)$. We say that this vertex is *bad* if it is incident to at least $n^{1-4\epsilon}$ edges of E_6, and that it is *good* otherwise. The following observation is central to dealing with the edges of E_6.

Observation 7.2 *Assume that (s,T) is a good pair, that all shortcut operations performed in previous phases and in Step 1 of the current phase were valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let $a^{\text{in}} \in V^{\text{in}} \setminus V(J)$ be a bad vertex. Then $a \in R$ must hold.*

We provide the proof of the observation below, after providing additional intuition for sparsification of the set E_6 of edges. Intuitively, we will set up an instance of the Directed Heavy Degree Estimation problem (see Definition 2.12), whose input is the graph H, the sets $Z = V^{\text{in}} \setminus V(J)$ and $Z' = V^{\text{out}} \setminus V(J)$ of vertices, threshold value $\tau = n^{1-4\epsilon}$, capacity thershold $c^* = M'$, and parameter $W = W_{\max}$. We will then use the algorithm from Claim 2.13 to this problem, and denote by $\hat{Y} \subseteq Z$ the outcome of the algorithm. Note that if the algorithm from Claim 2.13 does not err, then we are guaranteed that, for every vertex $x^{\text{in}} \in \hat{Y}$, at least $n^{1-4\epsilon}$ edges of E_6 are incident to x^{in}, so every vertex in $\hat{Y}$ is a bad vertex. Additionally, every vertex $a^{\text{in}} \in Z \setminus \hat{Y}$ is incident to at most $1000\tau \log n \cdot \log(W_{\max})$ edges of E_6. For every vertex $a^{\text{in}} \in \hat{Y}$, we will add a shortcut edge (a^{in}, t) to G'', H and H'; from Observation 7.2, if (s,T) is a good pair, all shortcut operations executed before Step 3 of the current phase have been valid, and Event $\hat{\mathcal{E}}$ did not happen, then this is a valid shortcut edge. We will then delete from H' all edges of E_6 leaving a^{in}. The edges of E_6 that are incident to the vertices of $Z \setminus \hat{Y}$ remain in H'; if the algorithm from Claim 2.13 does not err, their number is bounded by $O\left(n \cdot \tau \log n \cdot \log(W_{\max})\right) \leq O\left(n^{2-4\epsilon} \log n \cdot \log(W_{\max})\right)$. We now prove Observation 7.2.

Proof of Observation 7.2. Assume that (s, T) is a good pair, that all shortcut operations performed in previous phases and in Step 1 of the current phase are valid, and that Event $\hat{\mathcal{E}}$ did not happen. Let $a^{\text{in}} \in V^{\text{in}} \setminus V(J)$ be a bad vertex, and assume for contradiction that $a \notin R$, so $a \in L \cup S$.

We denote by $\hat{E}(a)$ the set of all edges of E_6 incident to a^{in}. Let $e = (a^{\text{in}}, v^{\text{out}}) \in \hat{E}(a)$ be any such edge. From the construction of the split graph, edge e does not lie in G'', so e is a backward edge corresponding to the edge $(v^{\text{out}}, a^{\text{in}})$ of G''. Since the current flow f is M'-integral, the flow on the edge $(v^{\text{out}}, a^{\text{in}})$ is at least M', and so the capacity of e in H is at least M'.

We further partition set $\hat{E}(a)$ into two subsets: set $\hat{E}'(a)$ containing all edges $(a^{\text{in}}, v^{\text{out}})$ where $v \in L$, and set $\hat{E}''(a)$ containing all remaining edges. From Inequality 3, $|L| \leq n^{\epsilon}$, and so $|\hat{E}''(a)| \geq n^{1-4\epsilon} - n^{\epsilon} \geq \frac{n^{1-4\epsilon}}{2}$.

Consider any edge $e = (a^{\text{in}}, v^{\text{out}}) \in \hat{E}''(a)$. From the definition of set $E''(a)$, $v \in S^{\text{out}} \cup R^{\text{out}}$, and from our assumption, $a^{\text{in}} \in S^{\text{in}} \cup L^{\text{in}}$. It is then immediate to verify that $e \in E'_S$ must hold (see Definition 6.1), and so $E''(a) \subseteq E'_S$.

Let $\hat{f}$ be a maximum s^{out}-t flow in H, so $\text{val}(\hat{f}) = F$, and let $\mathcal{P}$ be the flow-path decomposition of F. From Claim 6.3, $F = \hat{c}_S$ holds, and moreover, every path in $\mathcal{P}$ contains exactly one edge of E'_S. Since $E''(a) \subseteq E'_S$, we get that, for every edge $e \in E''(a)$, $\hat{f}(e) = \hat{c}(e)$ holds.

Recall that, from Observation 6.2, if we consider the partition (X, Y) of the vertices of H, where $X = L^* \cup S^{\text{in}}$, and $Y = V(H) \setminus X = S^{\text{out}} \cup R^* \cup \{t\}$, then $E_H(X, Y) = E'_S$. It then follows that, for every flow-path $P \in \mathcal{P}$, if we delete the unique edge of E'_S from P, then we obtain two subpaths of P: subpath P_1 whose vertices are contained in $L^* \cup S^{\text{in}}$, and subpath P_2, whose vertices are contained in $S^{\text{out}} \cup R^*$.

Let $\mathcal{P}' \subseteq \mathcal{P}$ be the set of paths $P \in \mathcal{P}$ with $E(P) \cap \hat{E}''(a) \neq \emptyset$. From our discussion, every path $P \in \mathcal{P}'$ contains exactly one edge of $\hat{E}''(a)$, and all vertices preceding this edge on the path lie in $L^* \cup S^{\text{in}}$.

Consider now any path $P \in \mathcal{P}'$, and let $v(P) \in V(P)$ be the first vertex of P that does not lie in $V(J)$. Since $a^{\text{in}} \notin V(J)$, $v(P)$ appears before a^{in} on P (or $v(P) = a^{\text{in}}$), and so $v(P) \in L^* \cup S^{\text{in}}$ must hold. We denote by $e(P)$ the edge of P immediately preceding $v(P)$ on the path. From our discussion so far:

$$\sum_{P \in \mathcal{P}'} \hat{f}(P) = \sum_{e \in \hat{E}''(a)} \hat{c}(e) \geq |E''(a)| \cdot M' \geq \frac{n^{1-4\epsilon} \cdot M'}{2}.$$

Recall that graph $\tilde{H}'$ was obtained from H by contracting all vertices of $V(H) \setminus V(J)$ into the vertex t'. Consider a path $P \in \mathcal{P}'$, and let P' be the subpath of P between s^{out} and $v(P)$. Let P'' be the path obtained from P' by replacing the last vertex $v(P)$ with t'. Note that P'' is a valid s^{out}-t' path in $\tilde{H}'$. From our discussion, path P' may not contain a vertex of $S^{\text{out}} \cup R^*$, and so it may not contain an edge of E'_S (see Definition 6.1). Therefore, path P'' must be problematic (see Definition 6.4). We define an s^{out}-t' flow f' in graph $\tilde{H}'$ as follows: for every path $P \in \mathcal{P}'$, we set $f'(P'') = \hat{f}(P)$. Notice that f' is a valid s^{out}-t' flow in $\tilde{H}'$, that sends at least $\sum_{P \in \mathcal{P}'} \hat{f}(P) \geq \frac{n^{1-4\epsilon} \cdot M'}{2}$ flow units via problematic paths. But from Corollary 6.6, if (s, T) is a good pair, all shortcut operations so far have been valid, and that Event $\hat{\mathcal{E}}$ did not happen, the total flow that f' may send via problematic paths is:

$$\sum_{P \in \mathcal{P}'} f'(P'') \leq 2^{14} n^{\epsilon} \cdot M' \cdot \gamma \cdot \log^2 n \cdot \log(W_{\max}) < \frac{n^{1-4\epsilon} \cdot M'}{2},$$

since $2^{20} \log^2 n \cdot \log(W_{\max}) \leq n^{\epsilon}$ from Inequality 5 and $\gamma < n^{1-6\epsilon}$ from Inequality 7, a contradiction.

We conclude that $a \in R$ must hold. □

Let $\tilde{\mathcal{E}}_i$ be the bad event that any of the shortcut operations performed in previous phases were invalid, and let $\tilde{\mathcal{E}}'_i$ be the bad event that any of the shortcut operations performed in Step 1 of the current phase was invalid. From Invariant I1, $\mathbf{Pr}\left[\tilde{\mathcal{E}}_i\right] \leq \frac{2i}{n^{\epsilon}\cdot\log^2(W_{\max})}$, and from Claim 5.8, $\mathbf{Pr}\left[\tilde{\mathcal{E}}'_i\right] \leq \frac{1}{n^{\epsilon}\cdot\log^2(W_{\max})}$.

7.2 Constructing the graph H'

We now provide an efficient algorithm for constructing the graph H'. We start with $V(H') = V(H)$ and $E(H') = \emptyset$, and then take care of the sets $E_2, \ldots, E_6$ of edges in turn.

Edges of E_2. Recall that set $E_2 \subseteq E(H)$ contains all special edges of H, and all edges of H that are incident to t'. We can compute the edges of E_2 by considering every vertex $v \in V(G)$ with $w(v) \geq M'$ in turn; for each such vertex v, we can use the data structure DS_f to determine the flow value $f(v^{\mathrm{in}}, v^{\mathrm{out}})$, which in turn allows us to determine whether the edges $(v^{\mathrm{in}}, v^{\mathrm{out}})$, $(v^{\mathrm{out}}, v^{\mathrm{in}})$ lie in H, and if so, to compute their residual capacity. It is then easy to see that set E_2 of edges can be computed in time $O(n)$; recall that $|E_2| \leq O(n)$ holds. We add the edges of E_2 to graph H'.

Edges of E_3. Next, we add all edges of E_3 to H'. Recall that set E_3 contains all regular edges (x, y) of H, such that either $x \in V(J)$, or at least one of the vertices x, y lies in Γ, excluding the edges that are incident to the vertices of $\left\{v^{\mathrm{in}}, v^{\mathrm{out}} \mid v \in V(G) \setminus U\right\}$. Recall that $|\Gamma| \leq \frac{32n^{1+2\epsilon}\cdot\log^2(W_{\max})}{\gamma}$. By inspecting the arrays corresponding to each of the vertices of Γ in the modified adjacency-list representation of H, we can compute all edges of E_3 that are incident to the vertices of Γ, in time $O(|\Gamma|\cdot n) \leq O\left(\frac{n^{2+2\epsilon}\cdot\log^2(W_{\max})}{\gamma}\right) \leq O(n^{2-4\epsilon})$ (we have used the fact that $\log^2(W_{\max}) \leq n^{\epsilon}$ from Inequality 5, and $\gamma \geq n^{9\epsilon}$ from Inequality 8). As observed already, if e is an edge of E_3 that is not incident to a vertex of Γ, then it must be the case that $e = (v^{\mathrm{in}}, u^{\mathrm{out}})$, where $v^{\mathrm{in}} \in V^{\mathrm{in}} \cap V(J)$ and $u^{\mathrm{out}} \in V^{\mathrm{out}} \setminus V(J)$, and moreover $e \in E^R$. For each such edge $e = (v^{\mathrm{in}}, u^{\mathrm{out}})$, the corresponding edge (v^{in}, t') lies in $\tilde{H}'$, and every edges incident to t' in $\tilde{H}'$ represents either an edge of E^R, or a special edge, so the number of such edges is bounded by $n + |E^R|$. We can then compute all edges of E^R by inspecting all edges that are incident to t' in $\tilde{H}'$, and considering, for each such edge e', the edge e of H that it represents. The time required to do so is bounded by $O(n + |E^R|) \leq O\left(n\cdot\gamma\cdot\log n\cdot\log(W_{\max})\right) \leq O(n^{2-4\epsilon})$, since $\gamma \leq n^{1-5\epsilon}$ from Inequality 7, and $\log n\cdot\log(W_{\max}) \leq n^{\epsilon}$ from Inequality 5; we have also used Property P4 of Observation 5.6 to bound $|E^R|$. Overall, we get that $|E_3| \leq O(|\Gamma|\cdot n) + O(|E^R|) \leq O(n^{2-4\epsilon})$, and our algorithm computes the set E_3 of edges in time $O(n^{2-4\epsilon})$. We add all edges of E_3 into H'.

Edges of E_4. Recall that set E_4 contains all regular edges (x, y) that do not lie in $E_1 \cup E_3$, such that $x \in V^{\mathrm{out}}$ and $y \in \left\{v^{\mathrm{in}} \mid v \in Q'\right\}$. Recall that we have set the value of δ so that $n^{\delta} = \frac{n^{4\epsilon}}{4000\log n}$ holds (see Equation 9), and from Inequality 10, $\delta \geq \frac{1}{\sqrt{\log n}}$. We perform a call to the δ-subgraph oracle with graph G and the set $Z = Q'$ of its vertices. Recall that the oracle must return a partition (Y^h, Y^{ℓ}) of $V(G)$, and a collection E' of at most $n^{2-\delta}\cdot\log^2(W_{\max}) = n^{2-4\epsilon}\cdot 4000\log n\cdot\log^2(W_{\max})$ edges of G, such that $E' = E_G(Y^{\ell}, Z)$ holds. Recall that, if the oracle does not err, then every vertex of Y^h has at least $\frac{n^{1-\delta}}{1000\log n} \geq 4n^{1-4\epsilon}$ neighbors in Q'. We let $\hat{\mathcal{E}}'$ be the bad event that the oracle erred. From the definition of the subgraph oracle, $\mathbf{Pr}\left[\hat{\mathcal{E}}'\right] \leq \frac{1}{n^5\cdot\log^4(W_{\max})}$. For every edge $(u, v) \in E'$ with $u \in U$, we add the corresponding edge $(u^{\mathrm{out}}, v^{\mathrm{in}})$, that must lie in E_4, to graph H'. Additionally, for every vertex $x \in Y^h$, we add the shortcut edge (x^{out}, t) to G'', H, and H'. We will use the following observation.

Observation 7.3 *If (s,T) is a good pair and neither of the events $\tilde{\mathcal{E}}_i, \tilde{\mathcal{E}}'_i, \hat{\mathcal{E}}, \hat{\mathcal{E}}'$ happened, then all shortcut operations performed while processing the set E_4 of edges are valid.*

Proof: Assume that (s,T) is a good pair, and that neither of the events $\tilde{\mathcal{E}}_i, \tilde{\mathcal{E}}'_i, \hat{\mathcal{E}}, \hat{\mathcal{E}}'$ has happened, so in particular, all shortcut operations prior to Step 3 of the current phase were valid. Then, from Observation 6.23 $|Q' \cap (S \cup L)| \leq 3n^{1-4\epsilon}$ holds. Consider now some vertex $x \in Y^h$. Since we have assumed that Event $\hat{\mathcal{E}}'$ did not happen, x has at least $4n^{1-4\epsilon}$ neighbors in Q' in graph G, and so in particular, x must have a neighbor that lies in R. We conclude that $x \in S \cup R$ must hold, and so the shortcut edge (x^{out}, t) is valid. □

To summarize, we add to H' all edges of E_4, except for the edges $(u^{\text{out}}, v^{\text{in}})$ with $u \in Y^h$. We also add, for every vertex $u \in Y^h$, the shortcut edge (u^{out}, t) to H'. The number of edges added to H' while processing E_4 is bounded by $O(|E'| + n) \leq O\left(n^{2-4\epsilon} \cdot \log n \cdot \log^2(W_{\max})\right)$. The time that the algorithm spends on processing the edges of E_4 (excluding the running time of the algorithm that implements the oracle) is bounded by $O(|E'| + n) \leq O\left(n^{2-4\epsilon} \cdot \log n \cdot \log^2(W_{\max})\right)$.

Edges of E_5. Recall that set E_5 contains all regular edges $(x,y) \in E(H)$ that do not lie in $E_1 \cup E_3 \cup E_4$, such that $y = v^{\text{in}}$ for some vertex $v \in Q$, and $x \in V^{\text{out}} \setminus V(J)$. If $|Q| > n^{1-4\epsilon}$, then we do not include any edges of E_5 in H'. Otherwise, we include all edges of E_5 in H'. In the latter case, in order to compute the edges of E_5, we simply inspect the arrays in the modified adjacency list representation of H that correspond to the vertices in $\left\{v^{\text{in}} \mid v \in Q\right\}$. Since $|Q| \leq n^{1-4\epsilon}$, we can compute the set E_5 of edges in time $O(|Q| \cdot n) \leq O(n^{2-4\epsilon})$, and $|E_5| \leq O(n^{2-4\epsilon})$ holds in this case.

Edges of E_6. Recall that, if $e = (x,y) \in E_6$, then, from Observation 7.1, $x \in V^{\text{in}} \setminus V(J)$ and $y \in V^{\text{out}} \setminus V(J)$ holds. We set up an instance of the Directed Heavy Degree Estimation problem (see Definition 2.12), whose input is the graph H, the sets $Z = V^{\text{in}} \setminus V(J)$ and $Z' = V^{\text{out}} \setminus V(J)$ of vertices, threshold value $\tau = n^{1-4\epsilon}$, capacity thershold $c^* = M'$, and parameter $W = W_{\max}$. We apply the algorithm from Claim 2.13 to this problem, and denote by $\hat{Y} \subseteq Z$ the outcome of the algorithm. Let $\hat{\mathcal{E}}''$ be the event that the algorithm from Claim 2.13 errs. Then, from Claim 2.13, $\mathbf{Pr}\left[\hat{\mathcal{E}}''\right] \leq \frac{1}{n^{10} \cdot \log^4(W_{\max})}$. If Event $\hat{\mathcal{E}}''$ did not happen, then for every vertex $x^{\text{in}} \in \hat{Y}$, at least $n^{1-4\epsilon}$ edges of E_6 are incident to x^{in}, so every vertex in $\hat{Y}$ is a bad vertex. Additionally, every vertex $a^{\text{in}} \in Z \setminus \hat{Y}$, is incident to at most $1000\tau \log n \cdot \log(W_{\max})$ edges of E_6. For every vertex $x^{\text{in}} \in \hat{Y}$, we add the shortcut edge (x^{in}, t) to G'', H, and H'. Note that, from Observation 7.2, if (s,T) is a good pair, and neither of the events $\tilde{\mathcal{E}}_i, \tilde{\mathcal{E}}'_i, \hat{\mathcal{E}}, \hat{\mathcal{E}}''$ happened, then for every vertex $x^{\text{in}} \in \hat{Y}$, $x \in R$ holds, and so all shortcut operations performed while processing the edges of E_6 are valid.

Next, we consider every vertex $x^{\text{in}} \in Z \setminus \hat{Y}$ in turn. As observed already, if Event $\hat{\mathcal{E}}''$ did not happen, then x^{in} is incident to at most $1000\tau \log n \cdot \log(W_{\max}) = 1000n^{1-4\epsilon} \log n \cdot \log(W_{\max})$ edges of E_6. Notice that the only regular edges that leave x^{in} are the edges of E_6, or the edges that are incident to vertices of $\Gamma = V^{\text{out}} \cap V(J)$ (this is since every edge $e \in E_6$ is a backward copy of a regular edge of G'', and so it may not lie in E_1). Since $|\Gamma| \leq N = \frac{32n^{1+2\epsilon} \cdot \log^2(W_{\max})}{\gamma} < n^{1-4\epsilon}$ (as $2^{10} \log n \cdot \log(W_{\max}) < n^{\epsilon}$ from Inequality 5 and $\gamma \geq n^{10\epsilon}$ from Inequality 8), we get that, if Event $\hat{\mathcal{E}}''$ did not happen, the total number of edges leaving the vertex x^{in} in H is bounded by $2000n^{1-4\epsilon} \log n \cdot \log(W_{\max})$. For every vertex $x^{\text{in}} \in Z \setminus \hat{Y}$, we process the lists of all edges leaving x^{in} in the modified adjacency-list representation of H. As we process these edges, we add each regular edge that is not incident to vertices of Γ to H'. If the number of such edges becomes greater than $1000n^{1-4\epsilon} \log n \cdot \log(W_{\max})$, then we terminate the algorithm and return "FAIL"; in this case, bad event $\hat{\mathcal{E}}''$ must have happened.

Recall that the running time of the algorithm for the Directed Heavy Degree Estimation problem from

Claim 2.13 is $O\left(\frac{n\cdot|Z'|\cdot\log n\cdot\log(W_{\max})}{\tau}\right) \leq O\left(n^{1+4\epsilon}\cdot\log n\cdot\log(W_{\max})\right) \leq O(n^{2-4\epsilon})$, since $\tau = n^{1-4\epsilon}$, $\log n\cdot\log(W_{\max}) \leq n^{\epsilon}$, and $\epsilon = \frac{1}{45}$. The remaining time required in order to recover the edges of E_6 that are incident to the vertices of $Z\setminus\hat{Y}$ is bounded by $O(|Z|\cdot n^{1-4\epsilon}\cdot\log n\cdot\log(W_{\max})) \leq O(n^{2-4\epsilon}\cdot\log n\cdot\log(W_{\max}))$. Note that we insert into H' all edges of E_6 that are incident to the vertices of $Z\setminus\hat{Y}$, and, for each vertex $x\in\hat{Y}$, we insert a shortcut edge (x,t) into H'.

This completes the construction of the graph H'. Consider now some edge $e\in E(H')$. If e is a regular edge, or a backward special edge, then we set its capacity $\hat{c}'(e)$ in H' to be $\hat{c}(e)$, the same as its capacity in H. Otherwise, e is a forward special edge, and we reduce its capacity by at least M' and at most $2M'$ units, to ensure that the resulting capacity $\max\{\hat{c}(e)-2M',0\} \leq \hat{c}'(e) \leq \hat{c}(e)-M'$ is an integral multiple of M' (recall that, from Property Q'1, $\hat{c}(e)\geq M'$ must hold). Since $W_{\max}, W'_{\max}$ and M' are all integral powers of 2, and since the current flow f is M'-integral, it is easy to verify that for every edge e of H', its capacity in H' is an integral multiple of M'. From the above discussion, it is immediate to verify that:

$$|E(H')| \leq O(n^{2-4\epsilon}\cdot\log n\cdot\log^3(W_{\max})), \tag{20}$$

and the time that our algorithm spends on constructing H' is bounded by $O(n^{2-4\epsilon}\cdot\log n\cdot\log^3(W_{\max}))$.

Recall that, if (s,T) is not a good pair, then any shortcut operation is valid by definition. From our discussion, if (s,T) is a good pair, and neither of the events $\tilde{\mathcal{E}}_i, \tilde{\mathcal{E}}'_i, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ happened, then all shortcut operations performed in Step 3 are valid ones.

Recall that $\tilde{\mathcal{E}}_i$ is the bad event that any of the shortcut operations performed in previous phases were invalid, and, from Invariant I1, $\mathbf{Pr}\left[\tilde{\mathcal{E}}_i\right] \leq \frac{2i}{n^{\epsilon}\cdot\log^2(W_{\max})}$. Recall that $\tilde{\mathcal{E}}'_i$ is the bad event that any of the shortcut operations performed in Step 1 of the current phase was invalid, and, from Claim 5.8, $\mathbf{Pr}\left[\tilde{\mathcal{E}}'_i\right] \leq \frac{1}{n^{\epsilon}\cdot\log^2(W_{\max})}$. Let $\tilde{\mathcal{E}}''_i$ be the bad event that at least one shortcut operation performed over the course of Step 3 is invalid. Notice that shortuct operations may only be performed during Step 3 when processing edges of E_4 or of E_6. From Observation 7.3, and our discussion above, if (s,T) is a good pair, and if neither of the events $\tilde{\mathcal{E}}_i, \tilde{\mathcal{E}}'_i$ has happened, then Event $\tilde{\mathcal{E}}''_i$ may only happen if either of the events $\hat{\mathcal{E}}, \hat{\mathcal{E}}'$, or $\hat{\mathcal{E}}''$ has happened. In other words, if (s,T) is a good pair, then:

$$\begin{aligned}\mathbf{Pr}\left[\tilde{\mathcal{E}}''_i \mid \neg\tilde{\mathcal{E}}_i\wedge\neg\tilde{\mathcal{E}}'_i\right] &\leq \mathbf{Pr}\left[\hat{\mathcal{E}}\vee\hat{\mathcal{E}}'\vee\hat{\mathcal{E}}'' \mid \neg\tilde{\mathcal{E}}_i\wedge\neg\tilde{\mathcal{E}}'_i\right]\\ &\leq \frac{1}{n^7\cdot\log^4(W_{\max})} + \frac{1}{n^5\cdot\log^4(W_{\max})} + \frac{1}{n^{10}\cdot\log^4(W_{\max})}\\ &\leq \frac{3}{n^5\cdot\log^4(W_{\max})}.\end{aligned}$$

(we have used Inequality 12 to bound $\mathbf{Pr}\left[\hat{\mathcal{E}}\right]$; the bounds on the probabilities of Events $\hat{\mathcal{E}}'$ and $\hat{\mathcal{E}}''$ follow from the above discussion.)

Let $\tilde{\mathcal{E}}_{i+1}$ be the bad event that any of the shortcut operations executed over the course of the first i phases was invalid. Then:

$$\begin{aligned}\mathbf{Pr}\left[\tilde{\mathcal{E}}_{i+1}\right] &\leq \mathbf{Pr}\left[\tilde{\mathcal{E}}_i\right] + \mathbf{Pr}\left[\tilde{\mathcal{E}}'_i\right] + \mathbf{Pr}\left[\tilde{\mathcal{E}}''_i \mid \neg\tilde{\mathcal{E}}_i \wedge \neg\tilde{\mathcal{E}}'_i\right] \\ &\leq \frac{2i}{n^{\epsilon}\cdot\log^2(W_{\max})} + \frac{1}{n^{\epsilon}\cdot\log^2(W_{\max})} + \frac{3}{n^5\cdot\log^4(W_{\max})} \\ &\leq \frac{2(i+1)}{n^{\epsilon}\cdot\log^2(W_{\max})}.\end{aligned} \tag{21}$$

Our algorithm will perform no further shortcut operations in the current phase, and so Invariant I1 holds at the end of the phase.

7.3 Properties of Graph H'

In this subsection we establish the main property of H', namely, that the value of the maximum s^{out}-t flow in H' is close to that in H, in the following claim.

Claim 7.4 *Let F be the value of the maximum s^{out}-t flow in H, and let F' be the value of the maximum s^{out}-t flow in H'. If $|Q| \leq n^{1-4\epsilon}$, then $F' \geq F - 2n\cdot M'$ holds. Additionally, if (s,T) is a good pair, and neither of the bad events $\mathcal{E}_i, \tilde{\mathcal{E}}_{i+1}, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ has happened, then $F' \geq F - 2n\cdot M' - 2n^{1-4.4\epsilon}\cdot M'$ holds even if $|Q| > n^{1-4\epsilon}$.*

Proof: We define an intermediate graph H'', that is obtained from H' by adding all edges of E_1 to it, and setting all edge capacities in H'' to be identical to those in H. Let F'' denote the value of the maximum s^{out}-t flow in H''. We use the following claim to lower-bound F''.

Claim 7.5 *If $|Q| \leq n^{1-4\epsilon}$, then $F'' = F$. Additionally, if (s,T) is a good pair and neither of the bad events $\mathcal{E}_i, \tilde{\mathcal{E}}_{i+1}, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ happened, then $F'' \geq F - 2n^{1-4.4\epsilon}\cdot M'$ even if $|Q| > n^{1-4\epsilon}$.*

Proof: Let f^* be the maximum s^{out}-t flow in H, and let $\mathcal{P}$ be its flow-path decomposition. Let $\mathcal{P}' \subseteq \mathcal{P}$ be the collection of all paths $P \in \mathcal{P}$ that are contained in H'', and let $\mathcal{P}'' = \mathcal{P} \setminus \mathcal{P}'$. Consider now a path $P \in \mathcal{P}''$, and let $e(P) \in E(P)$ be the first edge on P that does not lie in H''. Note that the edges of E_1, E_2 and E_3 are all present in H'', so $e(P) \in E_4 \cup E_5 \cup E_6$ must hold. We further partition the set $\mathcal{P}''$ into two subsets: set $\mathcal{P}''_1$ containing all paths $P \in \mathcal{P}''$ with $e(P) \in E_4 \cup E_6$, and set $\mathcal{P}''_2 = \mathcal{P}'' \setminus \mathcal{P}''_1$.

Consider first a path $P \in \mathcal{P}''_1$, and denote $e(P) = (x,y)$. Recall that $e(P) \in (E_4 \cup E_6) \setminus E(H'')$ holds. This can only happen if shortcut edge (x,t) was added to G'', H and H', so in particular this edge lies in H''. We let P' be the path obtained from P by contcatenating the subpath of P from s^{out} to x and the edge (x,t).

Let f' be the s^{out}-t flow in H'' obtained by sending, for every path $P \in \mathcal{P}'$, $f^*(P)$ flow units on P, and, for every path $P \in \mathcal{P}'_1$, $f^*(P)$ flow units on P'. Notice that f' is a valid s^{out}-t flow in H'', and its value is $\sum_{P\in\mathcal{P}'\cup\mathcal{P}''_1} f^*(P) = F - \sum_{P\in\mathcal{P}''_2} f^*(P)$. Observe that, if $|Q| \leq n^{1-4\epsilon}$, then all edges of E_5 were added to H', and so $\mathcal{P}''_2 = \emptyset$. Therefore, in this case, $F' \geq \sum_{P\in\mathcal{P}'\cup\mathcal{P}''_1} f^*(P) = F$.

Assume from now on that $|Q| > n^{1-4\epsilon}$, and additionally that (s,T) is a good pair and neither of the bad events $\mathcal{E}_i, \tilde{\mathcal{E}}_{i+1}, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ happened. It is now sufficient to prove that, in this case, $\sum_{P\in\mathcal{P}''_2} f^*(P) \leq 2n^{1-4.4\epsilon}\cdot M'$ holds.

Recall that for every edge $e = (x,y) \in E_5$, $y = v^{\text{in}}$ for some vertex $v \in Q$ and $x \in V^{\text{out}} \setminus V(J)$. In particular, $y \in X^*$ and $x \notin X^*$ holds. Since $s^{\text{out}} \in X^*$, we get that every path in $\mathcal{P}''_2$ must contain at

least two edges of $E_H(X^*, V(H) \setminus X^*)$. Clearly, every path in $\mathcal{P}$ must contain at least one such edge. Therefore:

$$\sum_{e \in E_H(X^*, V(H) \setminus X^*)} f^*(e) \geq F + \sum_{P \in \mathcal{P}_2''} f^*(P).$$

However, from Observation 6.23, $\sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e) \leq F + 2n^{1-4.4\epsilon} \cdot M'$ must hold, and so:

$$\sum_{e \in E_H(X^*, V(H) \setminus X^*)} f^*(e) \leq \sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e) \leq F + 2n^{1-4.4\epsilon} \cdot M'.$$

Altogether, we get that $\sum_{P \in \mathcal{P}_2''} f^*(P) \leq 2n^{1-4.4\epsilon} \cdot M'$, and so:

$$F'' \geq \sum_{P \in \mathcal{P}' \cup \mathcal{P}_1''} f^*(P) = F - \sum_{P \in \mathcal{P}_2''} f^*(P) \geq F - 2n^{1-4.4\epsilon} \cdot M'.$$

□

Observe that graph H' can be thought of as being obtained from H'' in two steps: first, we decrease the capacity of every forward special edge by at least M' and at most $2M'$ flow units. It is immediate to verify that this may only decrease the value of the maximum flow by at most $2nM'$ units, by inspecting the minimum s^{out}-t cut in H'', whose capacity may decrease by at most $2nM'$. Let $\hat{H}$ be the resulting flow network. Lastly, for every vertex $v \in V(G)$ with $w(v) < M'$, we delete all regular edges that are incident to v^{out} or v^{in} in $\hat{H}$, obtaining the final graph H'. We claim that this last step does not affect the value of the maximum s^{out}-t flow. Indeed, let $\hat{f}$ be the maximum s^{out}-t flow in $\hat{H}$, and denote its value by $\hat{F}$. Let $e = (v^{\text{in}}, v^{\text{out}})$ be a special edge corresponding to a vertex $v \in V(G)$ with $w(v) < M'$, and let $e' = (x, y)$ be a regular edge of H'' that is incident to either v^{in} or v^{out}. We claim that $\hat{f}(e') = 0$ must hold, so deleting this edge will not influence the flow $\hat{f}$. Indeed, recall that, from Property Q'1, $w(v) < M'$ must hold. Since the current flow f is M'-integral, $f(v^{\text{in}}, v^{\text{out}}) = 0$ must hold. Since the only edge leaving v^{in} in G'' is the edge $(v^{\text{in}}, v^{\text{out}})$, it is immediate to verify that, for every regular edge $(u^{\text{out}}, v^{\text{in}}) \in E(G'')$ that enters v^{in}, $f(u^{\text{out}}, v^{\text{in}}) = 0$. Therefore, the only regular edges incident to v^{in} in H are the forward regular edges that enter v^{in}. Since the capacity of the edge $(v^{\text{in}}, v^{\text{out}})$ in $\hat{H}$ is 0, we get that the flow $\hat{f}(e')$ on every edge $e' \in E(\hat{H})$ that is incident to v^{in} must be 0. Using a similar reasoning, the flow $\hat{f}(e'')$ on every edge $e'' \in E(\hat{H})$ that is incident to v^{out} must be 0. We conclude that $F' \geq \hat{F} \geq F'' - 2nM'$. From Claim 7.5, if $|Q| \leq n^{1-4\epsilon}$, then $F'' = F$, and so $F' \geq F - 2n \cdot M$. Moreover, if (s, T) is a good pair and if neither of the bad events $\mathcal{E}_i, \tilde{\mathcal{E}}_{i+1}, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ happened, then $F'' \geq F - 2n^{1-4.4\epsilon} \cdot M'$ holds even if $|Q| > n^{1-4\epsilon}$. In this case, $F' \geq F - 2nM' - 2n^{1-4.4\epsilon} \cdot M'$. □

7.4 Completing the Phase

In order to complete the algorithm for Step 3, we use the algorithm from Theorem 2.4 in order to compute a maximum s^{out}-t flow f'' in graph H'; since all edge capacities in H' are integral multiples of M', the resulting flow f'' is guaranteed to be M'-integral. The running time of the algorithm from Theorem 2.4 is $O\left(|E(H')|^{1+o(1)} \cdot \log(W_{\max})\right) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right)$, from Inequality 20.

Next, we consider two cases. The first case happens if $|Q| \leq n^{1-4\epsilon}$. In this case, we let f_i be the s^{out}-t flow in G'' obtained by augmenting the current flow f with the flow f''. From Claim 7.4, and because $F \geq \mathsf{OPT}_s - \text{val}(f)$ must hold, we get that:

$$\text{val}(f_i) = \text{val}(f) + \text{val}(f'') = \text{val}(f) + F' \geq \text{val}(f) + F - 2nM' \geq \mathsf{OPT}_s - 2nM'. \tag{22}$$

Note that the resulting flow f_i must be M'-integral. Given the flow f, the flow f_i can be computed in time $O(|E(H')|) \leq O(n^{2-4\epsilon} \cdot \log n \cdot \log^3(W_{\max}))$, and we can update the data structure DS_f, and the residual flow network H within the same asymptotic running time.

Assume now that $|Q| > n^{1-4\epsilon}$. In this case, we check whether $\mathrm{val}(f'') \geq \sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e) - 4n^{1-4.4\epsilon} \cdot M'$ holds, and, this is not the case, then we terminate the algorithm with a "FAIL". Notice that the time to compute the value $\sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e)$ is bounded by $O(|E(\tilde{H}')|)$, and it is subsumed by the running time of the algorithm for Step 3 so far. From Observation 6.23, if (s,T) is a good pair and neither of the bad events $\tilde{\mathcal{E}}_{i+1}$, $\hat{\mathcal{E}}$ happened, then $F \geq \sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e) - 2n^{1-4.4\epsilon} \cdot M'$. Additionally, from Claim 7.4, (s,T) is a good pair, and neither of the bad events $\mathcal{E}_i, \tilde{\mathcal{E}}_{i+1}, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ happened, then:

$$\mathrm{val}(f'') = F' \geq F - 2n \cdot M' - 2n^{1-4.4\epsilon} \cdot M' \geq \sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e) - 2n \cdot M' - 4n^{1-4.4\epsilon} \cdot M'.$$

Therefore, if $|Q| > n^{1-4\epsilon}$, (s,T) is a good pair, and neither of the bad events $\mathcal{E}_i, \tilde{\mathcal{E}}_{i+1}, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ has happened, we are guaranteed that $\mathrm{val}(f'') \geq \sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e) - 2n \cdot M' - 4n^{1-4.4\epsilon} \cdot M'$ must hold. So our algorithm may only terminate with a "FAIL" in this case if (s,T) is not a good pair, or at least one of the events $\mathcal{E}_i, \tilde{\mathcal{E}}_{i+1}, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ has happened.

Finally, assume that $|Q| > n^{1-4\epsilon}$, and that $\mathrm{val}(f'') \geq \sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e) - 4n^{1-4.4\epsilon} \cdot M'$. Clearly, $F \leq \sum_{e \in E_H(X^*, V(H) \setminus X^*)} \hat{c}(e)$ must hold. Therefore:

$$\mathrm{val}(f'') \geq F - 4n^{1-4.4\epsilon} \cdot M' - 2nM' \geq F - 4nM' \geq \mathsf{OPT}_s - \mathrm{val}(f) - 4nM'. \tag{23}$$

By augmenting the flow f with f'', exactly as before, we the obtain a new flow f_i, with $\mathrm{val}(f_i) = \mathrm{val}(f) + \mathrm{val}(f'') \geq \mathsf{OPT}_s - 4nM'$, that is M'-integral.

To summarize, the running time of the algorithm for Step 3 is bounded by $O(n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max}))$, and the total running time of the entire phase is bounded by $O(n^{2-4\epsilon+o(1)} \cdot (\log(W_{\max}))^{O(1)})$. If the algorithm for the ith phase does not terminate with a "FAIL", then it computes a flow f_i in G'', that is M'-integral, where $M' = M_i$, with $\mathrm{val}(f_i) \geq \mathsf{OPT}_s - 4nM' = \mathsf{OPT}_s - 4nM_i$, establishing Invariant I3. We have already established Invariant I1, and Invariant I5 follows from the fact that we decrease the residual capacity of every forward special edge in H' by at least $M' = M_i$ units. From Invariant I4, the total number of edges $e \in E(G'')$ with $f_{i-1}(e) > 0$ was at most $O\left(i \cdot n^{2-4\epsilon+o(1)} \cdot \log^4(W_{\max})\right)$. We only augmented the initial flow $f = f_{i-1}$ in Steps 1 and 3. Recall that the number of iterations in Step 1 is bounded by $z' = \frac{32n}{\gamma}$. In each iteration j, we may compute a flow $\hat{f}_j$ in the graph J that is constructed in the iteration, and then augment f with $\hat{f}_j$. Since, from Observation 5.3, $|E(J)| \leq O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right)$, we get that the total number of edges $e \in E(G'')$ with $f_{i-1}(e) = 0$, such that the flow $f(e) > 0$ holds at the end of Step 1 is bounded by:

$$O\left(\frac{n^{2+2\epsilon} \cdot \log^2(W_{\max})}{\gamma}\right) \cdot z' \leq O\left(\frac{n^{3+2\epsilon} \cdot \log^2(W_{\max})}{\gamma^2}\right) \leq O\left(n^{2-4\epsilon+o(1)} \cdot \log^2(W_{\max})\right),$$

since $\gamma \geq n^{2/3+5\epsilon}$ from Inequality 7. In Step 3 we perform a single augmentation of the flow f, by using the maximum s^{out}-t flow f'' in graph H'. Since $|E(H')| \leq O(n^{2-4\epsilon} \cdot \log n \cdot \log^3(W_{\max}))$ from Inequality 20, this augmentation may affect at most $O(n^{2-4\epsilon} \cdot \log n \cdot \log^3(W_{\max}))$ edges of G''. Overall, the number of edges $e \in E(G'')$ with $f_{i-1}(e) = 0$ and $f_i(e) > 0$ is bounded by $O\left(n^{2-4\epsilon+o(1)} \cdot \log^3(W_{\max})\right)$, and so Invariant I4 continues to hold. It now only remains to establish Invariant I2.

Assume that (s, T) is a good pair. The algorithm may terminate Phase i with a "FAIL" either in Step 1, or Step 2, or in Step 3. From Claim 5.8, if Event $\tilde{\mathcal{E}}_i$ did not happen, then the probability that the algorithm terminates with a "FAIL" in Step 1 of the current phase is at most $\frac{2}{n^\epsilon \cdot \log^2(W_{\max})}$. From Observation 6.15, the algorithm for Step 2 may only terminate with a "FAIL" if at least one of the events $\tilde{\mathcal{E}}_{i+1}$ or $\hat{\mathcal{E}}$ has happened. The algorithm may only terminate in Step 3 of the current phase with a "FAIL" if one of the bad events $\mathcal{E}_i, \tilde{\mathcal{E}}_{i+1}, \hat{\mathcal{E}}, \hat{\mathcal{E}}', \hat{\mathcal{E}}''$ happened. Overall, if (s, T) is a good pair, then the probability that the current phase terminates with a "FAIL" is bounded by:

$$\frac{2}{n^\epsilon \cdot \log^2(W_{\max})} + \mathbf{Pr}\left[\tilde{\mathcal{E}}_{i+1}\right] + \mathbf{Pr}\left[\hat{\mathcal{E}}\right] + \mathbf{Pr}\left[\hat{\mathcal{E}}'\right] + \mathbf{Pr}\left[\hat{\mathcal{E}}''\right].$$

Recall that $\mathbf{Pr}\left[\tilde{\mathcal{E}}_{i+1}\right] \leq \frac{2(i+1)}{n^\epsilon \cdot \log^2(W_{\max})}$ from Inequality 21, and $\mathbf{Pr}\left[\hat{\mathcal{E}}\right] \leq \frac{1}{n^7 \cdot \log^4(W_{\max})}$ from Inequality 12. Additionally, from our discussion, $\mathbf{Pr}\left[\hat{\mathcal{E}}'\right] \leq \frac{1}{n^5 \cdot \log^4(W_{\max})}$ and $\mathbf{Pr}\left[\hat{\mathcal{E}}''\right] \leq \frac{1}{n^{10} \cdot \log^4(W_{\max})}$. Altogether, we get that, if (s, T) is a good pair, then the probability that the current phase terminates with a "FAIL" is bounded by:

$$\begin{aligned}
&\frac{2}{n^\epsilon \cdot \log^2(W_{\max})} + \frac{2(i+1)}{n^\epsilon \cdot \log^2(W_{\max})} + \frac{1}{n^5 \cdot \log^4(W_{\max})} + \frac{1}{n^{10} \cdot \log^4(W_{\max})} \\
&\qquad \leq \frac{2i+5}{n^\epsilon \cdot \log^2(W_{\max})} \\
&\qquad \leq \frac{7i}{n^\epsilon \cdot \log^2(W_{\max})}.
\end{aligned}$$

From Invariant I2, if (s, T) is a good pair, then the probability that the algorithm terminates prior to the beginning of Phase i with a "FAIL" is at most $\frac{10i^2-2i}{n^\epsilon \cdot \log^2(W_{\max})}$. We then get that, if (s, T) is a good pair, then the probability that the algorithm terminates prior to the beginning of Phase i with a "FAIL" is at most $\frac{10i^2+5i}{n^\epsilon \cdot \log^2(W_{\max})} \leq \frac{10(i+1)^2-2(i+1)}{n^\epsilon \cdot \log^2(W_{\max})}$, so Invariant I2 continues to hold at the end of the phase.

Acknowledgement. The authors thank Ron Mosenzon for pointing out a minor error in the definition of a good set of terminals (Definition 3.12) in a previous version of the paper.

A Isolating Cuts: Proof of Theorem 2.7

In this section we provide the proof of Theorem 2.7. We restate this theorem for the convenience of the reader.

Theorem 2.7 (Isolating Cuts) *There is a deterministic algorithm, that, given as input an undirected m-edge graph G with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V(G)$, together with a subset $T \subseteq V(G)$ of vertices of G, computes, for every vertex $v \in T$, the value of the minimum vertex-cut separating v from $T \setminus \{v\}$. The running time of the algorithm is $O\left(m^{1+o(1)} \cdot \log W\right)$.*

In this proof, for a vertex set $X \subseteq V(G)$, we denote by $N_G(X) = \left(\bigcup_{v \in X} N_G(v)\right) \setminus X$. It will be convenient for us to assume that the vertices of T form an independent set, that is, no edge of G connects a pair of such vertices. In order to ensure this property, we simply subdivide every edge of G with a new vertex of weight $2nW$. It is easy to verify that, for every vertex $v \in T$, the value of the minimum vertex-cut separating v from $T \setminus \{v\}$ does not change following this transformation; if there is no vertex-cut separating v from $T \setminus \{v\}$ in G, then the value of the minimum vertex-cut separating v from $T \setminus \{v\}$ in the new graph is at least $2Wn$, higher than the value of any proper vertex-cut in G. Additionally, the number of edges in G only grows by factor 2. Therefore, we assume from now on that T is an independent set.

Let $r = \lceil \log |T| \rceil$. We assign, to every vertex $v \in T$, a binary string β_v of length r, so that strings assigned to distinct vertices of T are different. Next, we perform r iterations. For $1 \leq i \leq r$, in order to perform the ith iteration, we let $A_i \subseteq T$ contain all vertices $v \in T$ for which the ith bit in β_v is equal to 0, and we let $B_i = T \setminus A_i$. We then compute a minimum A_i-B_i vertex-cut (X_i, Y_i, Z_i) in G, in time $O\left(m^{1+o(1)} \cdot \log W\right)$, using the algorithm from Corollary 2.6. Consider now the graph $G \setminus (\bigcup_{i=1}^{r} Y_i)$, and, for every vertex $v \in T$, denote by C_v the connected component of this graph that contains v, by $U_v = V(C_v)$, and by $m_v = \sum_{u \in U_v} \deg_G(u)$. Observe that, for all $1 \leq i \leq r$, $Y_i \cap T = \emptyset$, and for all $v \in T$, $N_G(U_v) \subseteq \bigcup_{i=1}^{r} Y_i$. Therefore, for every vertex $v \in T$, $N_G(U_v) \cap T = \emptyset$. We start with the following simple observation.

Observation A.1 *For every vertex $v \in T$, $U_v \cap T = \{v\}$.*

Proof: Consider a vertex $v \in T$. By definition, $v \in U_v \cap T$. Assume for contradiction that U_v contains another vertex $u \neq v$ with $u \in T$. Since the binary strings assigned to u and v are different, there is an index $1 \leq j \leq r$, such that β_v and β_u differ in the jth bit. Assume without loss of generality that $u \in A_j$ and $v \in B_j$. Since $Y_j \subseteq V(G)$ separates A_j from B_j in G, every path P connecting u to v in G must contain a vertex of Y_j. Therefore, it is impossible that u and v lie in the same connected component of $G \setminus (\bigcup_{i=1}^{r} Y_i)$. □

From Observation A.1, we get that for every pair $u, v \in T$ of distinct vertices, $C_u \neq C_v$, and so $\sum_{v \in T} m_v \leq O(m)$.

For every vertex $v \in T$, we then construct a graph G_v as follows. We start from the subgraph of G induced by the set $U_v \cup N_G(U_v)$ of vertices, and then remove all edges $e = (x, y)$ with $x, y \in N_G(U_v)$. We then add a vertex t, and connect it to every vertex in $N_G(U_v)$ with an edge. This finishes the construction of the graph G_v. Note that $|E(G_v)| \leq O(m_v)$, and moreover, given the adjacency-list representaton of G, we can compute G_v in time $O(m_v)$, by inspecting the adjacency lists of the vertices of U_v. From our observation, since $(U_v \cup N_G(U_v)) \cap T = \{v\}$, we get that the $V(G_v) \cap T = \{v\}$. We use the algorithm from Corollary 2.6 to compute, in time $O\left(|E(G_v)|^{1+o(1)} \cdot \log W\right) \leq O\left(m_v^{1+o(1)} \cdot \log W\right)$, a minimum v-t vertex-cut in G_v, and denote by c_v its value. We then return, for every vertex $v \in T$, the value c_v as the value of the minimum vertex cut separating v from $T \setminus \{v\}$ in G. From our discussion, the

total running time of the algorithm is bounded by $O\left(m^{1+o(1)} \cdot \log W\right) + \sum_{v \in T} O\left(m_v^{1+o(1)} \cdot \log W\right) \leq O\left(m^{1+o(1)} \cdot \log W\right)$. It now remains to show that, for every vertex v, the value c_v that the algorithm outputs is indeed the value of the minimum vertex-cut separating v from $T \setminus \{v\}$ in G.

Consider any vertex $v \in T$, and let (L_v, S_v, R_v) be a minimum vertex-cut separating v from $T \setminus \{v\}$ in G, so that $v \in L_v$; from among all such cuts, choose the one minimizing $|L_v|$. We denote $w(S_v)$ by λ_v, so λ_v is the value of the minimum vertex-cut separating v from $T \setminus \{v\}$ in G. The following claim is key to the correctness of the algorithm.

Claim A.2 *For every vertex $v \in T$, $L_v \subseteq U_v$ must hold.*

We prove the claim below, after we complete the proof of Theorem 2.7 using it. Fix a vertex $v \in T$, and notice that, since $L_v \subseteq U_v$, $N_G(L_v) \subseteq U_v \cup N_G(U_v)$ must hold. Consider the partition (L', S', R') of the vertices of G_v, where $L' = L_v$; $S' = N_G(L_v)$; and R' contains the remaining vertices of G_v, so in particular $t \in R'$. Clearly, $S' \subseteq S_v$, and moreover, no edge of G_v may connect a vertex of L' to a vertex of R'. Therefore, (L', S', R') is a valid v-t vertex-cut in G_v, and its value is at most $\lambda_v = w(S_v)$. We conclude that $c_v \leq \lambda_v$. On the other hand, consider any v-t vertex-cut (A, B, C) in G_v, so $v \in A$ and $t \in C$ holds. Note that $A \subseteq U_v$ must hold, since every vertex $u \in V(G_v) \setminus \{t\}$ that does not lie in U_v is connected to t with an edge. In particular, this means that $N_G(A) = N_{G_v}(A) \subseteq B$. Consider now a partition (A', B', C') of $V(G)$, where $A' = A$, $B' = N_G(A)$, and C' contains the remaining vertices of G. From the above discussion, $B' \subseteq B$ must hold. Clearly, (A', B', C') is a vertex-cut in G, whose value is at most $w(B)$. Since $V(G_v) \cap T = \{v\}$, the only terminal that lies in $A \cup B$ is v, and so this cut separates v from $T \setminus \{v\}$. We conclude that $c_v \geq \lambda_v$, and altogether, $c_v = \lambda_v$. In order to complete the proof of Theorem 2.7 it now remains to prove Claim A.2.

Proof of Claim A.2. We fix a vertex $v \in T$. For all $1 \leq i \leq r$, we define a set Λ_i of vertices as follows. If $v \in A_i$, then we set $\Lambda_i = X_i$; note that $\Lambda_i \cap T = A_i$ in this case. Otherwise, $v \in B_i$, and we set $\Lambda_i = Z_i$; in this case, $\Lambda_i \cap T = B_i$. In the following claim we show that, for all $1 \leq i \leq r$, $L_v \subseteq \Lambda_i$.

Claim A.3 *For all $1 \leq i \leq r$, $L_v \subseteq \Lambda_i$.*

Proof: The proof of the claim relies on *submodularity of vertex-cuts*, that is summarized in the following observation.

Observation A.4 (Submodularity of weighted vertex-cuts.) *Let $G = (V, E)$ be an undirected graph with weights $w(v) \geq 0$ on its vertices $v \in V$, and let $A, B \subseteq V$ be a pair of vertex subsets. Then:*

$$w(N(A)) + w(N(B)) \geq w(N(A \cup B)) + w(N(A \cap B)).$$

Proof: We consider the contribution of every vertex $v \in V$ to the expressions:

$$w(N(A)) + w(N(B)) \tag{24}$$

and

$$w(N(A \cup B)) + w(N(A \cap B)). \tag{25}$$

Notice that a vertex $v \notin N(A) \cup N(B)$ contributes 0 to expression 24. We now show that it also contributes 0 to expression 25. Indeed, if no neighbor of v lies in $A \cup B$, then v may not lie in either

of the sets $N(A \cup B)$ or $N(A \cap B)$, so its contribution to expression 25 is 0. Assume now that some neighbor $u \in N(v)$ of v lies in A, but no neighbor of v lies in B. Since $v \notin N(A)$, $v \in A$ must hold. But then $v \in A \cup B$, and no neighbor of v may lie in $A \cap B$. Therefore, $v \notin N(A \cup B)$ and $v \notin N(A \cap B)$ must hold. The case where some neighbor of v lies in B but no neighbor of v lies in A is symmetric. Finally, assume that some neighbor of v lies in A, and the same holds for B. Since $v \notin N(A) \cup N(B)$, it must be the case that $v \in A$ and $v \in B$. But then $v \in A \cap B$, so it may not lie in either of the sets $N(A \cup B)$ or in $N(A \cap B)$. To conclude, if $v \notin N(A) \cup N(B)$, its contribution to both expressions is 0.

Assume now that $v \in N(A) \cup N(B)$. Observe first that, if $v \in N(A) \cap N(B)$, then it contributes $2w(v)$ to the first expression, and its contribution to the second expression is at most $2w(v)$.

Assume next that $v \in N(A) \setminus N(B)$, so v contributes $w(v)$ to expression 24. We show that in this case, either $v \notin N(A \cap B)$, or $v \notin N(A \cup B)$ must hold, so v contributes at most $w(v)$ to the expression 25. In order to do so, we consider two cases. The first case happens if $v \notin B$. In this case, no vertex of $N(v)$ may lie in B, so $v \notin N(A \cap B)$. Assume now that the second case happens, so $v \in B$. In this case, $v \notin N(A \cup B)$ must hold, since, from the definition, $N(A \cup B)$ does not contain vertices of $A \cup B$.

The only remaining case is where $v \in N(B) \setminus N(A)$, and it follows a symmetric argument. □

We are now ready to complete the proof of Claim A.3. Assume for contradiction that the claim does not hold, so $L_v \setminus \Lambda_i \neq \emptyset$. Consider a vertex-cut $(\hat{X}, \hat{Y}, \hat{Z})$ in G, where $\hat{X} = L_v \cap \Lambda_i$ and $\hat{Y} = N(L_v \cap \Lambda_i)$, and $\hat{Z}$ contains all remaining vertices of G. It is immediate to verify that this cut separates v from $T \setminus \{v\}$. Indeed, since $L_v \cap T = \{v\}$ and $v \in \Lambda_i$, we get that $\hat{X} \cap T = \{v\}$. Additionally, since $\hat{Y} \subseteq L_v \cup N(L_v)$ and $(L_v \cup N(L_v)) \cap T = \{v\}$, we get that $\hat{Y} \cap T = \emptyset$. Moreover, since $|\hat{X}| < |L_v|$, from the definition of the cut (L_v, S_v, R_v), we get that $w(\hat{Y}) > w(S_v)$ must hold. However, by Observation A.4:

$$w(N(L_v \cup \Lambda_i)) + w(N(L_v \cap \Lambda_i)) \leq w(N(L_v)) + w(N(\Lambda_i)). \tag{26}$$

Recall that (L_v, S_v, R_v) is a vertex-cut in G, so $N_G(L_v) \subseteq S_v$ must hold, and therefore, $w(N(L_v)) \leq w(S_v)$. We then get that:

$$w(N(L_v \cap \Lambda_i)) = w(N(\hat{X})) = w(\hat{Y}) > w(S_v) \geq w(N(L_v)).$$

Combining this with Inequality 26, we get that:

$$w(N(L_v \cup \Lambda_i)) < w(N(\Lambda_i))$$

must hold. Since $L_v \cap T = \{v\}$ and $v \in \Lambda_i$, we get that $(L_v \cup \Lambda_i) \cap T = \Lambda_i \cap T$. In particular, $L_v \cup \Lambda_i$ contains all vertices of A_i and no vertices of B_i (or the other way around). Moreover, since $N(L_v) \cap T = \emptyset$ and $N(\Lambda_i) \cap T = \emptyset$, we also get that $N(L_v \cup \Lambda_i) \cap T = \emptyset$. Therefore, the vertices of $N(L_v \cup \Lambda_i)$ separate the vertices of A_i from the vertices of B_i, and their weight is smaller than $w(N(\Lambda_i)) \leq w(Y_i)$, contradicting the choice of the vertex-cut (X_i, Y_i, Z_i). □

Let $U_v^* = \bigcap_{i=1}^{r} \Lambda_i$. We conclude that $L_v \subseteq U_v^*$. Recall that C_v is the connected component of $G \setminus (\bigcup_{i=1}^{r} Y_i)$ containing the vertex v. It is also easy to verify that $C_v \subseteq U_v^*$, and that equivalently, C_v can be defined as the connected component of $G[\bigcap_{i=1}^{r} \Lambda_i]$ containing v. Since $G[L_v]$ must be a connected graph from the minimality of $|L_v|$, it must be a subset of $U_v = V(C_v)$. □

B Degree Estimation and Heavy Vertex Problems

In this section we provide algorithms for both Undirected and Directed Degree Estimation problems, as well as an algorithm for the Heavy Vertex Problem. We start with an algorithm for the Undirected Degree Estimation problem.

B.1 Algorithms for Degree Estimation Problems: Proof of Claims 2.11 and 2.13

We start by presenting an algorithm for the Undirected Degree Estimation problem, proving Claim 2.11, whose statement we restate here for the convenience of the reader.

Claim 2.11 (Algorithm for the Undirected Degree Estimation Problem) *There is a randomized algorithm for the Undirected Degree Estimation problem, whose running time is* $O\left(\frac{n\cdot|Z|\cdot\log n\cdot\log(W_{\max})}{\tau}\right)$. *The probability that the algorithm errs is at most* $\frac{1}{n^{10}\cdot\log^6(W_{\max})}$, *where* $W_{\max}=W_{\max}(G)$.

Proof of Claim 2.11. Assume first that $\tau > \frac{|Z|}{100\log n\cdot\log(W_{\max})}$. In this case, we return $A=\emptyset$. It is easy to verify that in this case the algorithm does not err, since no vertex in Z' may have at least $1000\tau\log n\cdot\log(W_{\max}) > |Z|$ neighbors in Z. We assume from now on that $\tau \le \frac{|Z|}{100\log n\cdot\log(W_{\max})}$ holds.

We start by computing a random set $T\subseteq Z$ of vertices as follows: every vertex $v\in Z$ is added to T independently with probability $\frac{10\log n\cdot\log(W_{\max})}{\tau}$. Let $\mathcal{E}$ be the bad event that $|T| > \frac{100|Z|\log n\cdot\log(W_{\max})}{\tau}$. Since $\mathbf{E}\left[|T|\right]=\frac{10|Z|\log n\cdot\log(W_{\max})}{\tau}$, by applying the first Chernoff bound from Lemma 2.8 with $t=\frac{100|Z|\log n\cdot\log(W_{\max})}{\tau}$, we get that $\mathbf{Pr}\left[\mathcal{E}\right]\le 2^{-(100|Z|\log n\cdot\log(W_{\max}))/\tau}\le n^{-100\log(W_{\max})}$, since $\tau\le|Z|$. Notice that we can check whether Event $\mathcal{E}$ happened in time $O\left(\frac{|Z|\cdot\log n\log(W_{\max})}{\tau}\right)$. If we discover that the event happened, then we terminate the algorithm and return $A=\emptyset$.

Assume now that Event $\mathcal{E}$ did not happen. We let $A\subseteq Z'$ be the set that contains all vertices $v\in Z'$ with $|N_G(v)\cap T|\ge 100\log n\log(W_{\max})$. Note that set A of vertices can be computed in time $O(|T|\cdot n)\le O\left(\frac{n\cdot|Z|\cdot\log n\cdot\log(W_{\max})}{\tau}\right)$, as follows: for every vertex $u\in T$, we consider all its $O(n)$ neighbors in G. For each such neighbor v, if $v\notin Z'$, then no further processing of v is needed. Otherwise, if $v\in Z'$, and v is processed for the first time by our algorithm, then we initialize the counter $n_v=1$, and mark v as being already considered by the algorithm. We also add v to the list $\Lambda\subseteq Z'$ of vertices that our algorithm considered. Otherwise, we simply increase the counter n_v by 1. Clearly, the algorithm spends $O(n)$ time for processing every vertex $v\in T$, and $O(n|T|)\le O\left(\frac{n\cdot|Z|\cdot\log n\cdot\log(W_{\max})}{\tau}\right)$ time overall. We then consider the vertices $v\in\Lambda$ one by one, and for each such vertex v, if $n_v\ge 100\log n\cdot\log(W_{\max})$, then we add v to A.

Consider now some vertex $v\in Z'$. We say that a bad event $\mathcal{E}'(v)$ happened if $|N_G(v)\cap Z|<\tau$ and $v\in A$, and we say that a bad event $\mathcal{E}''(v)$ happened if $|N_G(v)\cap Z|\ge 1000\tau\log n\log(W_{\max})$ and $v\notin A$. We now bound the probabilities of events $\mathcal{E}'(v)$ and $\mathcal{E}''(v)$.

For convenience, denote $N_G(v)\cap Z=\{y_1,\ldots,y_r\}$. For all $1\le i\le r$, let x_i be the random variable that is equal to 1 if $y_i\in T$ and to 0 otherwise, and let $X_v=\sum_{i=1}^r x_i$. Notice that $\mathbf{E}\left[X_v\right]=\frac{10r\cdot\log n\cdot\log(W_{\max})}{\tau}$.

Assume first that $r<\tau$, so $\mathbf{E}\left[X_v\right]<10\log n\log(W_{\max})$, and recall that v is only added to A if $X_v>100\log n\cdot\log(W_{\max})$. In this case, by using the first part of the Chernoff bound from Lemma 2.8 with $t=100\log n\cdot\log(W_{\max})$, we get that:

$$\mathbf{Pr}\left[\mathcal{E}'(v)\right] = \mathbf{Pr}\left[X_v > 100\log n \cdot \log(W_{\max})\right] \leq 2^{-100\log n \cdot \log(W_{\max})} = n^{-100\cdot\log(W_{\max})}.$$

Assume now that $r \geq 1000\tau \log n \cdot \log(W_{\max})$, so $\mathbf{E}\left[X_v\right] \geq 1000\log^2 n \log^2(W_{\max})$. In this case, $\mathcal{E}''(v)$ may only happen if $X_v \leq 100\log n \cdot \log(W_{\max})$. Using the second part of the Chernoff bound from Lemma 2.8 with $\delta = 0.9$, we get that:

$$\begin{aligned}
\mathbf{Pr}\left[\mathcal{E}''(v)\right] &= \mathbf{Pr}\left[X_v < 100\log n \cdot \log(W_{\max})\right] \\
&\leq \mathbf{Pr}\left[X_v < (1-\delta)\cdot 1000\log^2 n \cdot \log^2(W_{\max})\right] \\
&\leq e^{-0.81\cdot 500\log^2 n \cdot \log^2(W_{\max})} \\
&\leq n^{-400\cdot\log(W_{\max})}.
\end{aligned}$$

Note that our algorithm may only err if the event $\mathcal{E}$ happens, or if, for some $v \in Z'$, at least one of the events $\mathcal{E}'(v)$ or $\mathcal{E}''(v)$ happen. Using the Union Bound, we conclude that the probability that the algorithm errs is at most

$$\begin{aligned}
&n^{-100\log(W_{\max})} + \sum_{v\in V}\left(n^{-100\cdot\log(W_{\max})} + n^{-400\cdot\log(W_{\max})}\right) \\
&\leq n^{-50\cdot\log(W_{\max})} \\
&\leq n^{-50-\log(W_{\max})} \\
&\leq \frac{1}{n^{10}\cdot\log^6(W_{\max})}.
\end{aligned}$$

This concludes the proof of Claim 2.11. We note that in the above proof we could have replaced $W_{\max}$ with any other parameter $\hat{W}$ that is greater than a large enough constant, so that $n^{\log(\hat{W})} \geq \log^6 \hat{W}$ holds.

We now restate Claim 2.13.

Claim 2.13 (Algorithm for the Directed Heavy Degree Estimation Problem) *There is a randomized algorithm for the Directed Heavy Degree Estimation problem, whose running time is $O\left(\frac{n\cdot|Z'|\cdot\log n\cdot\log W}{\tau}\right)$. The probability that the algorithm errs is at most $\frac{1}{n^{10}\cdot\log^6 W}$.*

The proof of Claim 2.13 easily follows by adapting the algorithm from the proof of Claim 2.11. The only difference is that we replace the parameter $W_{\max}$ with W and switch the roles of vertex sets Z and Z', so the set T of vertices is subsampled from Z'. We then add to the set A every vertex $v \in Z$, such that there are at least $100\log n \log W$ edges (v,u) with $u \in Z'$, whose capacity is at least c^*. The analysis of the algorithm remains essentially identical.

B.2 Algorithm for the Heavy Vertex Problem: Proof of Claim 2.15

In this section we provide a proof of Claim 2.15. First, we restate this claim for the convenience of the reader.

Claim 2.15 (Algorithm for the Heavy Vertex problem) *There is a randomized algorithm for the Heavy Vertex problem, whose running time, on input (G, Z, Z', τ, c^*, W), is $O\left(\frac{n\cdot|Z|\cdot\log n\cdot\log W}{\tau}\right)$. The probability that the algorithm errs is at most $1/\left(n^{10}\cdot\log^6 W\right)$.*

Proof of Claim 2.15. The algorithm closely follows the algorithm from the proof of Claim 2.13, except that we switch the roles of the sets Z and Z' of vertices, and reverse the directions of the edges. As before, if $\tau > \frac{|Z|}{100 \log n \cdot \log W}$, then we return $b = 0$; it is easy to verify that in this case the algorithm does not err. Otherwise, we select a subset $T \subseteq Z$ of vertices by adding every vertex $v \in Z$ to the set T independently with probability $\frac{10 \log n \cdot \log W}{\tau}$ as before. If $|T| \geq \frac{100|Z| \log n \cdot \log}{\tau}$, then we terminate the algorithm and return $b = 0$ as before. We assume from now on that $|T| < \frac{100|Z| \log n \cdot \log}{\tau}$. By inspecting the edges leaving the vertices of T, we identify a subset $A \subseteq Z'$, containing all vertices $v \in Z'$, such that the number of edges $e = (u, v)$ with $u \in T$ and $c(e) \geq c^*$ is at least $100 \log n \cdot \log W$. If $A = \emptyset$, then we return $b = 0$. Otherwise, we select an arbitrary vertex $v \in A$, and inspect all edges entering v in G, in time $O(n)$. If we find a set $E' \subseteq \delta^-(v)$ of at least τ edges (u, v) with $u \in Z$, whose capacities are at least c^*, then we select arbitrary τ such edges, and return $b = 1$ together with the endpoints of these edges that are distinct from v. Otherwise, we return $b = 0$. Following the same analysis as in the proofs of Claim 2.13 and Claim 2.11, it is easy to verify that the probability that the algorithm errs is at most $\frac{1}{n^{10} \cdot \log^6 W}$. Since $\tau \leq |Z|$, we get that $O(n) \leq O\left(\frac{n \cdot |Z| \cdot \log n \cdot \log W}{\tau}\right)$, so the running time of the algorithm remains bounded by $O\left(\frac{n \cdot |Z| \cdot \log n \cdot \log W}{\tau}\right)$.

References

[ACD+24] A. Aamand, J. Y. Chen, M. Dalirrooyfard, S. Mitrović, Y. Nevmyvaka, S. Silwal, and Y. Xu. Differentially private Gomory-Hu trees. *CoRR*, 2024. Available from: http://arxiv.org/abs/2408.01798.

[AKT21a] A. Abboud, R. Krauthgamer, and O. Trabelsi. APMF < APSP? Gomory-Hu tree for unweighted graphs in almost-quadratic time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1135–1146, 2021. doi:10.1109/FOCS52979.2021.00112.

[AKT21b] A. Abboud, R. Krauthgamer, and O. Trabelsi. Subcubic algorithms for Gomory-Hu tree in unweighted graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1725–1737, 2021. doi:10.1145/3406325.3451073.

[AKT22] A. Abboud, R. Krauthgamer, and O. Trabelsi. Friendly cut sparsifiers and faster Gomory-Hu trees. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 3630–3649, 2022. doi:10.1137/1.9781611977073.143.

[BBST24] A. Bernstein, J. Blikstad, T. Saranurak, and T.-W. Tu. Maximum flow by augmenting paths in $n^{2+o(1)}$ time. *CoRR*, 2024. Available from: https://arxiv.org/abs/2406.03648.

[BDD+82] M. Becker, W. Degenhardt, J. Doenhardt, S. Hertel, G. Kaninke, W. Kerber, K. Mehlhorn, S. Näher, H. Rohnert, and T. Winter. A probabilistic algorithm for vertex connectivity of graphs. *Inf. Process. Lett.*, 15(3):135–136, 1982. doi:10.1016/0020-0190(82)90046-1.

[BPWZ14] A. Björklund, R. Pagh, V. V. Williams, and U. Zwick. Listing triangles. In *International Colloquium on Automata, Languages, and Programming*, pages 223–234. Springer, 2014.

[CCPS21] R. Cen, Y. Cheng, D. Panigrahi, and K. Sun. Sparsification of Directed Graphs via Cut Balance. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198, pages 45:1–45:21, 2021. doi:10.4230/LIPIcs.ICALP.2021.45.

[CGK14] K. Censor-Hillel, M. Ghaffari, and F. Kuhn. A new perspective on vertex connectivity. In C. Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 546–561, 2014. doi:10.1137/1.9781611973402.41.

[CHI+17] S. Chechik, T. D. Hansen, G. F. Italiano, V. Loitzenbauer, and N. Parotsidis. Faster algorithms for computing maximal 2-connected subgraphs in sparse directed graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1900–1918, 2017. doi:10.1137/1.9781611974782.124.

[Chu16] J. Chuzhoy. Improved bounds for the excluded grid theorem. *arXiv preprint arXiv:1602.02629*, 2016. Available from: http://arxiv.org/abs/1602.02629.

[CKL+22] L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, and S. Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022. doi:10.1109/FOCS54457.2022.00064.

[CKT93] J. Cheriyan, M. Kao, and R. Thurimella. Scan-first search and sparse certificates: An improved parallel algorithms for k-vertex connectivity. *SIAM J. Comput.*, 22(1):157–174, 1993. doi:10.1137/0222013.

[CLN+21] R. Cen, J. Li, D. Nanongkai, D. Panigrahi, T. Saranurak, and K. Quanrud. Minimum cuts in directed graphs via partial sparsification. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1147–1158, 2021. doi:10.1109/FOCS52979.2021.00113.

[CLP22] R. Cen, J. Li, and D. Panigrahi. Augmenting edge connectivity via isolating cuts. In J. S. Naor and N. Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 3237–3252, 2022. doi:10.1137/1.9781611977073.127.

[CQ21] C. Chekuri and K. Quanrud. Faster algorithms for rooted connectivity in directed graphs. In N. Bansal, E. Merelli, and J. Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198, pages 49:1–49:16, 2021. doi:10.4230/LIPICS.ICALP.2021.49.

[CR94] J. Cheriyan and J. H. Reif. Directed *s-t* numberings, rubber bands, and testing digraph *k*-vertex connectivity. *Comb.*, 14(4):435–451, 1994. doi:10.1007/BF01302965.

[DP09] D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. Available from: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

[ET75] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975. doi:10.1137/0204043.

[Eve75] S. Even. An algorithm for determining whether the connectivity of a graph is at least k. *SIAM J. Comput.*, 4(3):393–396, 1975. doi:10.1137/0204034.

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956. Available from: http://www.rand.org/pubs/papers/P605/.

[FNY+20] S. Forster, D. Nanongkai, L. Yang, T. Saranurak, and S. Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2046–2065, 2020. doi:10.1137/1.9781611975994.126.

[FPZ23] K. Fox, D. Panigrahi, and F. Zhang. Minimum cut and minimum *k*-cut in hypergraphs via branching contractions. *ACM Trans. Algorithms*, 19(2):13:1–13:22, 2023. doi:10.1145/3570162.

[Gab95] H. N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995.

[Gab06] H. N. Gabow. Using expander graphs to find vertex connectivity. *J. ACM*, 53(5):800–844, 2006. doi:10.1145/1183907.1183912.

[GT88] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988. doi:10.1145/48014.61051.

[HHS24] Z. He, S. Huang, and T. Saranurak. Cactus representations in polylogarithmic max-flow via maximal isolating mincuts. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1465–1502, 2024. doi:10.1137/1.9781611977912.60.

[HLRW24] M. Henzinger, J. Li, S. Rao, and D. Wang. Deterministic near-linear time minimum cut in weighted graphs. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, pages 3089–3139, 2024. doi:10.1137/1.9781611977912.111.

[HO94] J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994. Announced at SODA 1992. doi:10.1006/JAGM.1994.1043.

[HRG00] M. R. Henzinger, S. Rao, and H. N. Gabow. Computing vertex connectivity: New bounds from old techniques. *J. Algorithms*, 34(2):222–250, 2000. Announced at FOCS 1996. doi:10.1006/JAGM.1999.1055.

[HRW17] M. Henzinger, S. Rao, and D. Wang. Local flow partitioning for faster edge connectivity. In P. N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1919–1938, 2017. doi:10.1137/1.9781611974782.125.

[Kan90] A. Kanevsky. On the number of minimum size separating vertex sets in a graph and how to find all of them. In D. S. Johnson, editor, *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1990*, pages 411–421, 1990. Available from: http://dl.acm.org/citation.cfm?id=320176.320227.

[Kar00] D. R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. Announced at STOC 1996. doi:10.1145/331605.331608.

[Kle69] D. J. Kleitman. Methods for investigating connectivity of large graphs. *IEEE Transactions on Circuit Theory*, 16(2):232–233, 1969. doi:10.1109/TCT.1969.1082941.

[KP15] D. Kang and J. Payor. Flow rounding. *CoRR*, 2015. Available from: https://arxiv.org/pdf/1507.08139.

[KS93] D. R. Karger and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In S. R. Kosaraju, D. S. Johnson, and A. Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993*, pages 757–765, 1993. doi:10.1145/167088.167281.

[KT19] K. Kawarabayashi and M. Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019. doi:10.1145/3274663.

[Li21] J. Li. Deterministic mincut in almost-linear time. In S. Khuller and V. V. Williams, editors, *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 384–395, 2021. doi:10.1145/3406325.3451114.

[LLW88] N. Linial, L. Lovász, and A. Wigderson. Rubber bands, convex embeddings and graph connectivity. *Combinatorica*, 8(1):91–102, 1988. doi:10.1007/BF02122557.

[LNP+21] J. Li, D. Nanongkai, D. Panigrahi, T. Saranurak, and S. Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 317–329, 2021. doi:10.1145/3406325.3451088.

[LNPS23] J. Li, D. Nanongkai, D. Panigrahi, and T. Saranurak. Near-linear time approximations for cut problems via fair cuts. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 240–275, 2023. doi:10.1137/1.9781611977554.CH10.

[LP20] J. Li and D. Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 85–92, 2020. doi:10.1109/FOCS46700.2020.00017.

[LP21] J. Li and D. Panigrahi. Approximate Gomory-Hu tree is faster than $n-1$ max-flows. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1738–1748. ACM, 2021. doi:10.1145/3406325.3451112.

[LPS21] J. Li, D. Panigrahi, and T. Saranurak. A nearly optimal all-pairs min-cuts algorithm in simple graphs. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1124–1134, 2021. doi:10.1109/FOCS52979.2021.00111.

[Men27] K. Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927. Available from: http://eudml.org/doc/211191.

[MN21] S. Mukhopadhyay and D. Nanongkai. A note on isolating cut lemma for submodular function minimization. *arXiv preprint arXiv:2103.15724*, 2021.

[NI92] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

[NSY19] D. Nanongkai, T. Saranurak, and S. Yingchareonthawornchai. Breaking quadratic time for small vertex connectivity and an approximation scheme. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 241–252, 2019. doi:10.1145/3313276.3316394.

[NW61] C. S. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 01 1961. doi:10.1112/jlms/s1-36.1.445.

[Pod73] V. D. Podderyugin. An algorithm for finding the edge connectity of graphs, 1973. Available from: https://cir.nii.ac.jp/crid/1371694371159352077.

[ST83] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

[SY22] T. Saranurak and S. Yingchareonthawornchai. Deterministic small vertex connectivity in almost linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 789–800, 2022. doi:10.1109/FOCS54457.2022.00080.

[Tut61] W. T. Tutte. On the problem of decomposing a graph into n connected factors. *Journal of the London Mathematical Society*, s1-36(1):221–230, 01 1961. doi:10.1112/jlms/s1-36.1.221.

[vdBCP+23] J. van den Brand, L. Chen, R. Peng, R. Kyng, Y. P. Liu, M. P. Gutenberg, S. Sachdeva, and A. Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 503–514, 2023. doi:10.1109/FOCS57990.2023.00037.

[WXXZ24] V. V. Williams, Y. Xu, Z. Xu, and R. Zhou. New bounds for matrix multiplication: from alpha to omega. In D. P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, pages 3792–3835, 2024. doi:10.1137/1.9781611977912.134.

[Zha22] T. Zhang. Faster cut-equivalent trees in simple graphs. In M. Bojanczyk, E. Merelli, and D. P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 109:1–109:18, 2022. doi:10.4230/LIPICS.ICALP.2022.109.