Faster Algorithms for Global Minimum Vertex-Cut in Directed Graphs*

Julia Chuzhoy† Ron Mosenzon‡ Ohad Trabelsi§

Abstract. We study the directed global minimum vertex-cut problem: given a directed vertex-weighted graph G, compute a vertex-cut (L, S, R) in G of minimum value, which is defined to be the total weight of all vertices in S. The problem, together with its edge-based variant, is one of the most basic in graph theory and algorithms, and has been studied extensively. The fastest currently known algorithm for directed global minimum vertex-cut (Henzinger, Rao and Gabow, FOCS 1996 and J. Algorithms 2000) has running time $\tilde{O}(mn)$, where m and n denote the number of edges and vertices in the input graph, respectively. A long line of work over the past decades led to faster algorithms for other main versions of the problem, including the undirected edge-based setting (Karger, STOC 1996 and J. ACM 2000), directed edge-based setting (Cen et al., FOCS 2021), and undirected vertex-based setting (Chuzhoy and Trabelsi, STOC 2025). However, for the vertex-based version in directed graphs, the 29 year-old $\tilde{O}(mn)$-time algorithm of Henzinger, Rao and Gabow remains the state of the art to this day, in all edge-density regimes.

In this paper we break the $\Theta(mn)$ running time barrier for the first time, by providing a randomized algorithm for directed global minimum vertex-cut, with running time $O\left(mn^{0.976} \cdot \operatorname{poly}\log W\right)$ where W is the ratio of largest to smallest vertex weight. Our algorithm can also be viewed as improving and significantly simplifying the recent randomized $O(mn^{0.99+o(1)} \cdot \operatorname{poly}\log W)$-time algorithm for the undirected version of the problem (Chuzhoy and Trabelsi, STOC 2025).

Additionally, we provide a randomized $O\left(\min\left\{m^{1+o(1)} \cdot k, n^{2+o(1)}\right\}\right)$-time algorithm for the unweighted version of directed global minimum vertex-cut, where k is the value of the optimal solution. The best previous algorithms for the problem achieved running time $\tilde{O}\left(\min\left\{k^2 \cdot m, mn^{11/12+o(1)}, n^{2+o(1)}\right\}\right)$ (Forster et al., SODA 2020) and (Li et al., STOC 2021). Our result almost matches the best current running time of $\tilde{O}\left(\min\left\{k \cdot m, n^2\right\}\right)$ for the directed unweighted edge-based version of the problem, due to (Gabow, STOC 1991 and J. Comput. Syst. Scie., 95) and (Chekuri, Quanrud, ICALP 2021).

1 Introduction

Given a directed graph $G = (V, E)$ with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V$, a *vertex-cut* is a partition (L, S, R) of V into three disjoint subsets with $L, R \neq \emptyset$, such that no edge connects a vertex of L to a vertex of R (but edges connecting vertices of R to vertices of L are allowed). The *value* of the cut is $w(S)$ – the total weight of all vertices in S. In the global minimum vertex-cut problem, the goal is to compute a vertex-cut of minimum value in the input directed vertex-weighted graph G. We also consider the *unweighted* version of the problem, where the weights of all vertices are unit, and the value of the vertex-cut (L, S, R) is $|S|$. In this paper we explore the design of fast algorithms for directed global minimum vertex-cut in both the unweighted and the general weighted settings. A closely related and extensively studied variant of the problem is based on edge-cuts. In this variant, the weights are on the graph's edges instead of vertices, and an edge-cut is a bipartition (L, R) of $V(G)$ into non-empty subsets. The *value* of the cut is the total weight of all edges connecting vertices of L to vertices of R in the weighted version of the problem, or just the number $|E_G(L, R)|$ of such edges in the unweighted version. In the global minimum edge-cut problem, the goal is to

†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: ron.mosenzon@ttic.edu. Supported in part by NSF grant CCF-2402283.

§University of Haifa, Israel. Email: otrabelsi@cs.haifa.ac.il. Work mostly done while at Toyota Technological Institute at Chicago.

compute an edge-cut of minimum value in the given input graph. Throughout, we use m and n to denote the number of edges and vertices, respectively, in the input graph G, and W to denote the maximum vertex weight.

Edge- and vertex-cuts in graphs, together with the closely related notion of flows, are among the most fundamental graph-theoretic objects, that arise routinely in algorithms and structural results involving graphs. The unweighted version of global min-cut is closely related to another central graph-theoretic notion: vertex-connectivity (or edge-connectivity in the case of edge-cuts). It is then not surprising that the design of fast algorithms for global minimum cut has been studied extensively over many decades. In fact one of the most well-known results in the area of graph optimization is the edge contraction based algorithm of Karger and Stein [21] for the edge version of the problem in undirected graphs.

Clearly, the value of the global minimum cut in a graph is equal to the smallest value, over all pairs (s,t) of its vertices, of the minimum s-t cut, in both the edge- and the vertex-versions of the problem. A central paradigm in designing efficient algorithms for global minimum cut is to identify a small number of such pairs (s,t), so that computing a minimum s-t cut for each such pair is sufficient in order to find the global min-cut. In order to optimize the running time, algorithms for computing a minimum cut separating a subset S of vertices of G from some vertex t, that we refer to as the minimum S-t cut, are also sometimes utilized. The classical algorithm of Hao and Orlin [14] for global minimum edge-cut in directed graphs provides an ingenious implementation of this paradigm. Their algorithm uses a variant of the Push-Relabel scheme of [13] in order to compute a minimum S-t cut for some initial set S of vertices and a vertex $t \notin S$. The inner state of the Push-Relabel algorithm is then exploited in order to identify a new set S' of vertices and a new vertex $t' \notin S'$, for which the minimum S'-t' cut can be computed efficiently by building on the current inner state of the Push-Relabel algorithm. After $O(n)$ such iterations, the algorithm is guaranteed to terminate with a global minimum edge-cut, in time $\tilde{O}(mn)$. Henzinger et al. [17] extended this method to the vertex-weighted version of the problem in directed graphs, obtaining a randomized $\tilde{O}(mn)$-time algorithm for directed global minimum vertex-cut. This remains the fastest current algorithm for the problem in every edge-density regime, although its running time was very recently almost matched, to within an $n^{o(1)}$ factor, by a deterministic algorithm [18].

For the edge version of the problem, significantly faster algorithms have been known for some time. In undirected graphs, Karger [20], improving upon the algorithm of [21] mentioned above, provided a randomized algorithm with close to the best possible running time of $\tilde{O}(m)$; a recent sequence of works [16, 22, 24, 15] has led to a deterministic algorithm with a similar running time. In directed graphs, the best current algorithm for global minimum edge-cut, due to Cen et al. [4] solves the problem in the time of $\tilde{O}\left(\min\left\{\frac{n}{m^{1/3}}, \sqrt{n}\right\}\right)$ applications of Maximum s-t Flow, that, combined with the recent almost-linear time algorithm for the latter problem by [7, 36], yields a randomized algorithm with $O\left(\min\left\{\frac{n}{m^{1/3}}, \sqrt{n}\right\} \cdot m^{1+o(1)}\right)$ running time for directed edge-weighted global minimum cut. The edge version of the problem was also studied extensively in directed unweighted graphs [9, 32, 28, 12, 6], with the best current algorithms achieving running time $\tilde{O}(mk)$ where k is the value of the optimal solution [12] and $\tilde{O}(n^2)$ [6].

Despite receiving a significant amount of attention starting from as early as the late 1960's [23, 31, 9, 27], the progress on the vertex version of the problem has been slower, and the state of the art results are significantly weaker. One exception is the **unweighted undirected** version, where a recent series of results [29, 11] culminated with an $O\left(m^{1+o(1)}\right)$-time randomized algorithm [25]; this bound relies on the recent almost-linear time algorithm for directed Maximum s-t Flow [7, 36]. A very recent work of [18] provides a deterministic algorithm for this version of the problem, with running time $O(m^{1+o(1)} \cdot k)$, where k is the size of the global min-cut. For **weighted undirected** graphs, the $\Theta(mn)$ barrier of [17] on the running time was broken only recently by [8], who obtained a randomized algorithm with running time $O(\min\{mn^{0.99+o(1)}, m^{3/2+o(1)}\} \cdot \operatorname{poly}\log W)$. For **unweighted directed** graphs, the best current algorithms, due to [11, 25], have running time $\tilde{O}\left(\min\left\{k^2 \cdot m, mn^{11/12+o(1)}, n^{2+o(1)}\right\}\right)$, when combined with the almost-linear time algorithm for Maximum s-t Flow of [7, 36]. For **weighted directed** graphs, the 29 year-old randomized algorithm of [17] with $\tilde{O}(mn)$ running time remains the fastest known to this day in every edge-density regime; a very recent result of [18] provided a deterministic algorithm with an almost matching running time $O(mn^{1+o(1)})$.

The directed vertex-version of the problem appears to be especially difficult, because some of the powerful techniques that were pivotal to obtaining fast algorithms in other settings no longer seem to apply in this setting. This includes, for example, the *tree packing* method of Tutte and Nash-Williams [35, 30], that is used

by most algorithms for the edge-cut version, and the *Isolating Cuts* lemma of [26, 1], that is used in algorithms for undirected vertex-weighted graphs. Additionally, a standard reduction from directed vertex-capacitated to directed edge-capacitated graphs via *split graphs* (see the definition in Section 2.4), that works in the context of Maximum s-t Flow, does not appear to work in the context of global minimum vertex-cut (see the discussion in the introduction of [8]), and so it is unclear whether the recent faster algorithms for directed global minimum edge-cut of [4] can be leveraged for the vertex-cut version.

In this paper we break the $\Theta(mn)$ running time barrier for directed global minimum vertex-cut, by designing a randomized algorithm with running time $O\left(mn^{0.976} \cdot \operatorname{poly}\log W\right)$. Our algorithm can also be viewed as improving and simplifying the recent $O\left(\min\{mn^{0.99+o(1)}, m^{3/2+o(1)}\} \cdot \operatorname{poly}\log W\right)$-time algorithm of [8] for undirected global minimum vertex-cut. Additionally, we provide a new randomized $O\left(\min\left\{m^{1+o(1)} \cdot k, n^{2+o(1)}\right\}\right)$-time algorithm for unweighted global minimum vertex-cut in directed graphs, where k is the value of the optimal solution. This running time almost matches the $\tilde{O}\left(\min\left\{k \cdot m, n^2\right\}\right)$-time algorithm for the unweighted edge version of the problem in directed graphs of [12, 6]. The best previous algorithm for the unweighted directed global minimum vertex-cut achieved running time $\tilde{O}(\min\{k^2 \cdot m, mn^{11/12+o(1)}, n^{2+o(1)}\})$ [11, 25].

In order to put our results for directed vertex-weighted graphs into context, we provide a very high-level overview of the recent $O(mn^{0.99+o(1)} \cdot \operatorname{poly}\log W)$-time algorithm of [8] for the undirected version of the problem. At a very high level, [8] provide two simple algorithms for two special cases of the problem. The first special case is where there exists an optimal vertex-cut (L, S, R) that is roughly balanced (that is, $|L|, |R| \geq n^{\epsilon}$ for some small constant ϵ). The algorithm for this special case is very simple and works as is in directed graphs; we employ it as well in this special case. The second special case is where every global minimum vertex-cut is unbalanced but the graph is not too dense (e.g. $m \leq n^{1.98}$). Their algorithm for this special case is also simple, but it crucially relies on the powerful Isolating Cuts lemma of [26, 1], that does not have known analogues in directed graphs. One of the main difficulties in designing a fast algorithm for directed global minimum vertex-cut is overcoming this obstacle, and we do so in this paper. Finally, [8] provide a very technically involved algorithm for the remaining special case, where the input graph is very dense, and every optimal vertex-cut is unbalanced. We provide an algorithm for this special case in directed graphs, that we believe is significantly simpler, and that achieves a faster running time. Overall, our technical contribution to the weighted version of the problem is twofold. First, we provide a randomized algorithm with running time $O\left(mn^{11/12+o(1)} \cdot d^{1/12} \cdot \operatorname{poly}\log W\right)$, where d is the average vertex degree. This algorithm allows us to handle graphs that are not too dense, a task that was relatively easy for the undirected setting, but is much more challenging in directed graphs. This algorithm is completely new. Second, we provide an $O\left(n^{2.677} \cdot \operatorname{poly}\log W\right)$-time algorithm for directed global minimum vertex-cut, that can be used in the case where the input graph is dense. This algorithm builds on and generalizes the algorithm of [8] to the directed setting, while also improving its running time and significantly simplifying it.

Independent work. Independently from our work, [10] showed a simple reduction from directed to undirected vertex connectivity in dense graphs. In particular, their results lead to a much simpler $n^{2+o(1)}$-time algorithm for the directed unweighted Global Minimum Vertex-Cut, than the algorithm of [25]. They also obtain a subcubic-time algorithm, with running time $O(n^{2.99+o(1)})$ for weighted directed Global Minimum Vertex-Cut by combining this reduction with the algorithm of [8] for undirected Global Minimum Vertex-Cut.

Other related work. A thorough and extensive overview of past results for global min-cut and the related techniques can be found in [8]. We only mention here several additional recent developments on directed vertex-weighted global minimum cut, that include a $(1+\varepsilon)$-approximation algorithm with running time $\tilde{O}(n^2/\varepsilon^2)$ by [4], and an exact algorithm with pseudo-polynomial running time $\tilde{O}(\mathsf{OPT} \cdot n \cdot \sum_{u \in V} w(u))$ by [6].

1.1 Our Results and Techniques

Note that it is possible that a directed graph G contains no vertex-cuts, for example, if the out-degree of every vertex in G is $n-1$. Given a graph G, we can check this in time $O(|E(G)|)$. Therefore, in the statements of our results, we assume that the input graph G has a valid vertex-cut.

Our main result is a new algorithm for the global minimum vertex-cut problem in directed graphs, that is summarized in the following theorem.

THEOREM 1.1. *There is a randomized algorithm, that, given a simple directed n-vertex and m-edge graph G with*

integral weights $0 \leq w(v) \leq W$ *on its vertices* $v \in V(G)$ *that contains some vertex-cut, computes a vertex-cut* (L', S', R') *in* G*, such that, with probability at least* $\frac{1}{2}$*,* (L', S', R') *is a global minimum vertex-cut. The running time of the algorithm is:*

$$O\left(\min\left\{mn^{11/12+o(1)} \cdot d^{1/12}, n^{2.677}\right\} \cdot (\log W)^{O(1)}\right) \leq O\left(mn^{0.976} \cdot (\log W)^{O(1)}\right),$$

where d *is the average vertex degree in* G*.*

The best previous randomized algorithm for the problem achieved running time $\tilde{O}(mn)$ [17], and a deterministic algorithm with an almost matching $O(mn^{1+o(1)})$ running time was recently provided by [18]. We also note that the best current algorithm for the undirected version of the problem has running time $O(\min\{mn^{0.99+o(1)}, m^{3/2+o(1)}\} \cdot \operatorname{poly}\log W)$ [8]; our result can be viewed as improving and simplifying this algorithm.

Our second result is a new algorithm for the unweighted directed global minimum vertex-cut problem, that is summarized in the following theorem.

THEOREM 1.2. *There is a randomized algorithm, that, given a simple directed unweighted* n*-vertex and* m*-edge graph* G *that contains some vertex-cut, computes a vertex-cut* (L', S', R') *in* G*, such that, with probability at least* $\frac{1}{2}$*,* (L', S', R') *is a global minimum vertex-cut. The expected running time of the algorithm is:*

$$O\left(\min\left\{m^{1+o(1)} \cdot k, n^{2+o(1)}\right\}\right),$$

where k *is the value of the global minimum vertex-cut in* G*.*

The best previous algorithms for the unweighted version of the problem achieve running times $\tilde{O}(k^2 m)$ [11] and $O(\min\{mn^{11/12+o(1)}, n^{2+o(1)}\})$ [25] (the latter runtime bound relies on the recent almost linear time algorithm for maximum s-t flow by [7, 36]). Both these algorithms are randomized. We note that the running time of our algorithm almost matches the $\tilde{O}\left(\min\left\{k \cdot m, n^2\right\}\right)$-time bound of the algorithm for the unweighted edge version of the problem in directed graphs of [12, 6].

We now provide a high-level overview of our techniques, starting with the algorithm for the unweighted version of the problem, which is simpler.

Unweighted Directed Graphs: Proof of Theorem 1.2 We assume that we are given as input a directed unweighted m-edge and n-vertex graph G, that contains a vertex-cut, and we denote by k the value of the global minimum vertex-cut in G. Our main technical subroutine is an algorithm that, given a "guess" $1 \leq \hat{k} \leq n/10$ on the value of the global minimum vertex-cut in G, produces some vertex-cut $(\hat{L}, \hat{S}, \hat{R})$ in G, in time $O\left(m^{1+o(1)} \cdot \hat{k}\right)$. The subroutine further guarantees that, if $\frac{\hat{k}}{2} \leq k \leq \hat{k}$, then, with a sufficiently high probability, $(\hat{L}, \hat{S}, \hat{R})$ is a global minimum vertex-cut in G. We execute this subroutine in parallel for $O(\log n)$ "guesses" $\hat{k}$ that are integral powers of 2 between 1 and $n/10$, and combine it with the $O\left(n^{2+o(1)}\right)$-time randomized algorithm of [25] in a natural way, in order to complete the proof of Theorem 1.2. From now on we focus on the description of this subroutine.

For the sake of the analysis, we fix an arbitrary global minimum vertex-cut (L, S, R) in G that we refer to as the *distinguished cut*. We can assume without loss of generality that $|L| \leq |R|$ holds, since otherwise we can instead focus on solving the problem on the reversed graph $\overline{G}$, that is obtained from G by reversing the direction of all its edges. We also assume without loss of generality that we are given a guess λ on $\mathrm{Vol}^+(L)$, that is an itegeral power of 2, such that $\frac{\lambda}{2} \leq \mathrm{Vol}^+(L) \leq \lambda$ holds; here, $\mathrm{Vol}^+(L)$ is the total out-degree of all vertices in L. In fact, our algorithm will try all such values of λ and then return the smallest-value vertex-cut computed using any such value λ.

A natural approach for solving the problem, that was also employed by [25] in the context of undirected graphs, is as follows. We compute a set $T \subseteq V(G)$ of vertices called *terminals*, by adding every vertex $v \in V(G)$ to T independently with probability roughly $\frac{\deg^+(v)}{\lambda}$, where $\deg^+(v)$ is the out-degree of v. We also select an additional vertex $t \in V(G)$ uniformly at random. We say that a good event $\mathcal{E}$ happens if all of the following hold: (i) $t \in R$;

(ii) $T \cap L \neq \emptyset$; and (iii) $|T| \leq \tilde{O}(m/\lambda)$. It is not hard to show that $\mathbf{Pr}[\mathcal{E}] \geq \Omega\left(\frac{1}{\log n}\right)$; we assume for simplicity in the remainder of the exposition that Event $\mathcal{E}$ has happened. For every vertex $v \in T$, we can then compute the value c_v of the minimum v-t vertex-cut in G. Let $v^* \in T$ be the vertex for which c_{v^*} is minimized. Then we compute the minimum v^*-t vertex-cut in G using the almost linear-time algorithm for maximum s-t flow of [7, 36], and output this vertex-cut as the algorithm's outcome. Since we have assumed that $t \in R$ and that $T \cap L \neq \emptyset$, it is easy to see that the resulting cut must be a global minimum vertex-cut. The main challenge in employing this approach is that $|T|$ may be quite large, especially if $\mathrm{Vol}^+(L)$ is small. Therefore, in order to achieve a running time that is below $\Theta(mn)$, we cannot afford to spend much time in order to compute the value of the minimum v-t vertex-cut in G for every vertex $v \in T$; we may need to perform each such computation in time that is significantly lower than m – the size of the input graph G. In order to overcome this difficulty, [25] design an ingenious and very efficient algorithm that, for every vertex $v \in T$, computes a *sparsified* subgraph $G_v \subseteq G$, such that, if $v \in L$ and $t \in R$, then the value of the minimum v-t cut in G_v is preserved, and otherwise it does not decrease. Computing the minimum v-t cut in each resulting graph G_v is then sufficient in order to complete the algorithm. Unfortunately, their sparsification procedure appears tailored to the undirected setting, and it is not clear how to generalize it to directed graphs. To summarize, the main difficulty with employing the above approach in directed graphs is that, since $|T|$ may be very large, it is not clear how to compute the value of the minimum v-t vertex-cut for all vertices $v \in T$ efficiently.

Our main idea for overcoming this difficulty can be thought of as *committing to the terminals gradually*. Specifically, we use a parameter $z = O(\log n)$, and we compute a hierarchical partition $(\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_z)$ of the terminals. We ensure that $\mathcal{B}_0 = \{T\}$, and, for all $1 \leq i \leq z$, $\mathcal{B}_i$ is a partition of the set T of terminals into roughly 2^i subsets of roughly equal cardinality; we refer to the subsets $B \in \mathcal{B}_i$ as *level-i batches*. We also ensure that every level-z batch $B \in \mathcal{B}_z$ contains exactly one terminal, and that, for all $1 \leq i \leq z$, partition $\mathcal{B}_i$ is a refinement of the partition $\mathcal{B}_{i-1}$ of T, so that every level-i batch is contained in some level-$(i-1)$ batch. At a high level, our algorithm consists of z phases. For all $1 \leq i \leq z$, in the ith phase, we compute, for every level-i batch $B \in \mathcal{B}_i$, a "sparsified" graph G_B, that has the following properties. First, $V(G_B) \subseteq V(G)$, $t \in V(G_B)$, and $|E(G_B)| \leq \tilde{O}\left(\frac{m \cdot \hat{k}}{2^i}\right)$ must hold. Additionally, we ensure that, for every vertex-cut (L', S', R') in G_B with $t \in R'$, $(L', S', V(G) \setminus (L' \cup S'))$ is a valid vertex-cut in G. Finally, we ensure that, if $B \cap L \neq \emptyset$, then $L \cup S \subseteq V(G_B)$, and, moreover, $(L, S, V(G_B) \setminus (L \cup S))$ is a valid vertex-cut in G_B.

Recall that every level-z batch $B \in \mathcal{B}_z$ contains a single terminal. Since we have assumed that $T \cap L \neq \emptyset$, there must be a level-z batch $B^* \in \mathcal{B}_z$, whose unique terminal $v^* \in B^*$ lies in L. We are then guaranteed that there is a vertex-cut (L', S', R') in G_{B^*} with $v^* \in L'$ and $t \in R'$, whose value is k; this is precisely the cut $(L, S, V(G_B) \setminus (L \cup S))$. For every level-$z$ batch $B \in \mathcal{B}_z$, we compute a minimum vertex-cut in G_B separating the unique terminal of B from t, and we let (L'', S'', R'') be the smallest-value resulting cut, whose value must be k from the above discussion. We then output the vertex-cut $(L'', S'', V(G) \setminus (L'' \cup S''))$ in G as the algorithm's outcome. Note that our algorithm ensures that, for every level $0 \leq i \leq z$, the total size of all graphs in $\{G_B \mid B \in \mathcal{B}_i\}$ is bounded by $\tilde{O}(m \cdot \hat{k})$. In particular, we can perform a single s-t vertex-cut computation in every graph G_B for $B \in \mathcal{B}_z$, in total time $O\left(m^{1+o(1)} \cdot \hat{k}\right)$.

We now provide additional technical details of our algorithm. Recall that there is a single level-0 batch of terminals, $B = T$. We let the corresponding graph G_B be G. It is immediate to verify that this graph has all required properties. Consider now some level $1 \leq i < z$, and some level-i batch $B \in \mathcal{B}_i$. Assume that we have computed the corresponding graph G_B with the required properties. For every level-$(i+1)$ batch $B' \subseteq B$, we construct the corresponding graph $G_{B'}$ as follows. First, we carefully compute a vertex-cut (L', S', R') in G_B with $t \in R'$, such that $|S'| \leq O\left(\frac{\hat{k} \cdot |T|}{2^i}\right)$, and, moreover, if $L \cap B' \neq \emptyset$, then $L \subseteq L'$ holds. This cut is computed by performing a single s-t vertex-cut computation in a graph $\hat{G}_B$ that is obtained by a slight modification of the graph G_B. We then let the new graph $G_{B'}$ contain the vertex set $L' \cup N^+_G(L') \cup \{t\}$. For every vertex $v \in L'$, we include all edges of $\delta^+_G(v)$ in $G_{B'}$, and for every vertex $v' \in N^+_G(L')$, we include the edge (v', t). Recall that, if $B' \cap L \neq \emptyset$, then our algorithm guarantees that $L \subseteq L'$ must hold. In this case, it is immediate to verify that $L \cup S \subseteq V(G_B)$, and, moreover, that $(L, S, V(G_B) \setminus (L \cup S))$ is a valid vertex-cut in $G_{B'}$. It is also not hard to see that, for every vertex-cut (L'', S'', R'') in $G_{B'}$ with $t \in R''$, $(L'', S'', V(G) \setminus (L'' \cup S''))$ is a valid vertex-cut in G. The main challenge however is to ensure that $|E(G_{B'})|$ is sufficiently small; since $|E(G_{B'})| = \Theta(\mathrm{Vol}^+_G(L'))$,

equivalently, it is enough to ensure that $\mathrm{Vol}^+_G(L')$ is sufficiently small.

In order to overcome this difficulty, at the beginning of the algorithm, in addition to selecting the set T of terminals, we also select a set $T' \subseteq V(G)$ of vertices that we refer to as *anti-terminals*, using a similar procedure. We say that a good event $\mathcal{E}'$ happens if Event $\mathcal{E}$ happened, and, additionally, $T' \cap L = \emptyset$. It is not hard to show that $\mathbf{Pr}[\mathcal{E}'] \geq \Omega(1/\operatorname{poly}\log n)$. Whenever we process a level-i batch B and its level-$(i+1)$ child-batch $B' \subseteq B$ as described above, we will ensure that the vertex-cut (L', S', R') in G_B is selected so that $L' \cap T' = \emptyset$. Recall that the cut (L', S', R') defines a valid vertex-cut (L', S', R'') in graph G, where $R'' = V(G) \setminus (L' \cup S')$. The key to the analysis of our algorithm is to show that, for every vertex-cut (L', S', R'') in G that may arise via this process, $\mathrm{Vol}^+_G(L')$ must be sufficiently small. This property follows from the selection of the set T' of the anti-terminals, and since, for each vertex-cut (L', S', R'') in G that may arise over the course of the algorithm, $L' \cap T' = \emptyset$ must hold.

Weighted Graphs: Proof of Theorem 1.2 We now provide a high-level description of our algorithm for weighted graphs, that is used in the proof of Theorem 1.1. Recall that we are given as input a directed n-vertex and m-edge graph $G = (V, E)$ with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V$; in fact, using a simple standard transformation described in Subsection 2.3, we can assume that $w(v) \geq 1$ for all $v \in V$, which we do in the remainder of this exposition. For a subset $X \subseteq V$ of vertices, we denote by $w(X) = \sum_{v \in X} w(v)$ its *weight*, and by $\mathrm{Vol}(X) = \sum_{v \in X} \deg_G(v)$ its *volume*. We also denote by $\mathrm{Vol}^+(X)$ the total out-degree of all vertices in X. As before, we can assume without loss of generality that there is a global minimum vertex-cut (L, S, R) in G with $w(L) \leq w(R)$; if this is not the case, we can equivalently solve the problem on the reversed graph $\overline{G}$, obtained from G by reversing the directions of all is edges. As before, for the sake of the analysis, we fix a single global minimum vertex-cut (L, S, R), that we refer to as the *distinguished cut* in the remainder of this exposition, to be any minimum vertex-cut with $w(L) \leq w(R)$. Throughout, we denote by d the average vertex degree in G and by OPT the value of the global minimum vertex-cut in G. We design two separate algorithms: one, with running time $O\left(mn^{11/12+o(1)} \cdot d^{1/12} \cdot \operatorname{poly}\log W\right)$, that will be employed when the input graph G is not too dense, and one with running time $O\left(n^{2.677} \cdot \operatorname{poly}\log W\right)$ that will be used in dense graphs. The proof of Theorem 1.1 follows immediately by combining these two algorithms. We now describe each of these two algorithms in turn.

Weighted Non-Dense Graphs We now describe our $O\left(mn^{11/12+o(1)} \cdot d^{1/12} \cdot \operatorname{poly}\log W\right)$-time algorithm, which is most useful when the input graph G is not too dense. We note that [8] provide a rather simple algorithm for the undirected setting that beats the $\Theta(mn)$ running time barrier if the input graph is not too dense. Their algorithm relies on the Isolating Cuts lemma of [26, 1]. Unfortunately, this technique is not known to extend to directed graph, and so designing an algorithm with running time below $\Theta(mn)$ for directed non-dense graphs appears to be a challenge in its own right.

We use a parameter $0 \leq \epsilon \leq 1$, that will be set to $\frac{1}{12} - \frac{\log d}{12 \log n}$. As in much of previous work, we design a separate algorithm to deal with several simple special cases, that we discuss next.

Simple special cases. Consider the distinguished min-cut (L, S, R). We say that it is *balanced* if either $|L| \geq n^\epsilon$ or $\mathrm{Vol}(L) \geq d \cdot n^\epsilon$ hold, and we say that it is *unbalanced* otherwise. For the case where cut (L, S, R) is balanced, it is easy to obtain a randomized algorithm with $O(mn^{1-\epsilon+o(1)} \cdot \log W)$ running time, using techniques similar to those employed in previous work, such as, e.g. [8]. For example, assume that $|L| \geq n^\epsilon$, and consider selecting a vertex s from $V(G)$ uniformly at random, and then selecting another vertex t from $V(G) \setminus (\{s\} \cup N^+(s))$ with probability proportional to its weight. Then it is not hard to show that, with probability at least $\Omega\left(\frac{1}{n^{1-\epsilon}}\right)$, $s \in L$ and $t \in R$ must hold. We can then compute a minimum s-t vertex-cut in G using the almost linear-time algorithm of [7, 36]. By repeating this process $\Theta(n^{1-\epsilon} \cdot \log n)$ times and returning the smallest-value cut among the resulting vertex-cuts, we are guaranteed to obtain a global minimum vertex-cut with probability at least 0.99; the running time of this algorithm is $O(mn^{1-\epsilon+o(1)} \cdot \log W)$. In the case where $\mathrm{Vol}(L) \geq d \cdot n^\epsilon$, we can employ a similar algorithm, except that we select the vertex s slightly differently: we first select an edge e from $E(G)$ uniformly at random, and then let s be its random endpoint. The remainder of the algorithm remains unchanged, and is guaranteed to return a global minimum vertex-cut with probability at least 0.99, in time $O(mn^{1-\epsilon+o(1)} \cdot \log W)$. To summarize, the special case where the distinguished min-cut is balanced is easy to solve using known techniques, so from now on we focus on the case where (L, S, R) is unbalanced, so $|L| \leq n^\epsilon$ and $\mathrm{Vol}(L) \leq d \cdot n^\epsilon$ hold. Since

every vertex of S must have a neighbor in L (as the vertex-cut (L,S,R) could be improved otherwise), we get that $|S| \leq d \cdot n^{\epsilon}$ must hold as well in this case. It will be convenient for us to assume that the algorithm is given an approximate estimate λ (to within factor 2) on $|L|$. In our actual algorithm we will try all $O(\log n)$ such possible estimates that are integral powers of 2. Clearly, it is enough to ensure that the vertex-cut that the algorithm returns is optimal with a sufficiently high probability, if the estimate on $|L|$ that it receives is approximately correct. Therefore, we assume from now on that the algorithm is given a value λ with $\lambda/2 \leq |L| \leq \lambda$, and that the distinguished cut is unbalanced, so $|L| \leq n^{\epsilon}$, $\mathrm{Vol}(L) \leq n^{\epsilon} \cdot d$ and $|S| \leq n^{\epsilon} \cdot d$ hold. We say that a vertex $v \in V$ is a *low-degree* vertex if $\deg(v) \leq d \cdot n^{\epsilon}$, and we say that it is a *high-degree* vertex otherwise. Note that, from the above discussion, L may not contain high-degree vertices. We now describe the main tools that our algorithm uses, that include a critical threshold, β-sets, σ-sets, and suspicious vertices.

Critical threshold. One of the central notions that our algorithm uses is that of a *critical threshold*. Consider some value $\tau \geq 1$ that is an integral power of 2. We say that τ is the *critical threshold* for the distinguished cut (L,S,R), if every **low-degree** vertex $v \in S$ has $w(v) \leq \tau$, and, moreover, τ is the smallest integral power of 2 with this property. It will be convenient for us to assume that our algorithm is given as input the value τ of the critical threshold for (L,S,R). In fact our algorithm will try all $O(\log(W))$ such values, and it is sufficient for us to ensure that the vertex-cut that the algorithm returns is optimal (with a sufficiently high probability), if the guessed value τ is indeed equal to the critical threshold. Therefore, we assume from now on that the critical threshold τ is known to the algorithm.

β-sets. The second main tool that we use is vertex sets $\beta(v)$ for all $v \in V(G)$, that are constructed as follows. Consider a vertex $v \in V(G)$, and consider performing a DFS search in G starting from v, while only exploring low-degree vertices whose weight is greater than τ. Once the DFS search discovers $|L|+1$ vertices, we terminate it (unless it terminated earlier), and we let $\beta(v)$ be the set of all vertices that the DFS has discovered, including v. A central observation is that, if $v \in L$, then $\beta(v) \subseteq L$ must hold. Indeed, otherwise, $\beta(v)$ must contain a vertex of S. But since all vertices in $\beta(v) \setminus \{v\}$ are low-degree vertices of weight greater than τ, from the definition of the critical threshold this is impossible. Therefore, if $v \in L$, then $\beta(v) \subseteq L$ and $N^{+}(\beta(v)) \subseteq L \cup S$ must hold. We will use this fact later.

σ-sets and suspicious vertices. For every vertex $u \in V(G)$, we then define the vertex set $\sigma(u) = \{v \in V(G) \mid u \in \beta(v)\}$. Note that, from the definition, $\sum_{v \in V(G)} |\beta(v)| \leq 2n|L|$, so on average, $|\sigma(u)| \leq 2|L|$ must hold. We say that a vertex u is *suspicious*, if $|\sigma(u)| > 2|L| \cdot n^{\epsilon}$, and we denote by U the set of all suspicious vertices in G; it is easy to verify that $|U| \leq n^{1-\epsilon}$. Intuitively, the algorithm that we present below for weighted non-dense graphs only works efficiently if no vertex of L is suspicious. However, the special case where $U \cap L \neq \emptyset$ is easy to deal with, via an algorithm that is very similar to the one that we used for the case where the cut (L,S,R) is balanced: we consider all suspicious vertices one by one. For each such vertex $s \in U$, we select a vertex t from $V(G)$ using a simple random process that ensures that, if $s \in L$, then, with a constant probability, $t \in R$ must hold. We then compute a minimum s-t vertex-cut in G using the almost linear-time algorithm of [7, 36]. Finally, we return the smallest-value vertex-cut among all resulting cuts. Like in the algorithm for the case where the cut (L,S,R) is balanced, we are guaranteed to obtain a global minimum vertex-cut with a sufficiently high probability, in time $O(mn^{1-\epsilon+o(1)} \cdot \log W)$. Therefore, we assume from now on that no vertex of L is suspicious. In particular, if $v \in L$, then no vertex of $\beta(v)$ may be suspicious, since, as observed already, $\beta(v) \subseteq L$ must hold.

We are now ready to describe our algorithm for the weighted non-dense setting. We select a set Γ of $\Theta(n/|L|)$ pairs of vertices of G using a simple randomized procedure that ensures that, with a sufficiently high probability, there is some pair $(x^*,y^*) \in \Gamma$, that we refer to as the *distinguished pair*, with $x^* \in L$ and $y^* \in R$. We can also ensure that, for every pair $(x,y) \in \Gamma$, x is a low-degree vertex and $(x,y) \notin E(G)$. In the remainder of the algorithm, our goal is to compute, for every pair $(x,y) \in \Gamma$, a value $c_{x,y}$ that is at least as high as the value of the minimum x-y vertex-cut in G; additionally, if $(x,y) \in \Gamma$ is the distinguished pair, then we need to ensure that $c_{x,y}$ is equal to the value of the minimum x-y vertex-cut in G. Once we compute the values $c_{x,y}$ for all pairs $(x,y) \in \Gamma$, we will select a pair $(x',y') \in \Gamma$ for which the value $c_{x',y'}$ is the smallest, and compute the minimum x'-y' vertex-cut in G, that is guaranteed to be a global minimum vertex-cut with a sufficiently high probability.

Consider now any such pair $(x,y) \in \Gamma$. In order to compute the value $c_{x,y}$ with the desired properties, our

main strategy is to compute a relatively small subset $A_{x,y} \subseteq V(G)$ of vertices, such that all vertices in $A_{x,y}$ are low-degree, $x \in A_{x,y}$ and $y \notin A_{x,y} \cup N^+(A_{x,y})$. Additionally, we will ensure that, if (x,y) is the distinguished pair, then $L \subseteq A_{x,y}$ holds. Assume first that we can indeed compute the vertex set $A_{x,y}$ with these properties, and let $A'_{x,y} \subseteq V(G)$ contain all vertices $v \in V(G) \setminus A_{x,y}$, such that there is an edge connecting some vertex of $A_{x,y}$ to v in G. Note that, if (x,y) is the distinguished pair, then $L \cup S \subseteq A_{x,y} \cup A'_{x,y}$. Consider now the graph $\hat{G}_{x,y}$, whose vertex set is $A_{x,y} \cup A'_{x,y} \cup \{\tilde{t}\}$, and the weights of the vertices in $A_{x,y} \cup A'_{x,y}$ remain the same as in G. For every vertex $v \in A_{x,y}$, we include in $\hat{G}_{x,y}$ all edges leaving v in G. For every vertex $u \in A'_{x,y}$, we include the edge $(u,\tilde{t})$ in $\hat{G}_{x,y}$. Since $|A_{x,y}|$ is relatively small, and all vertices in $A_{x,y}$ are low-degree vertices, $|E(\hat{G})|$ is relatively small as well. We can then afford to compute a minimum x-$\tilde{t}$ vertex-cut $(\hat{L},\hat{S},\hat{R})$ in $\hat{G}_{x,y}$. It is not hard to verify that the tripartition (L',S',R') of vertices of G, where $L' = \hat{L}$, $S' = \hat{S}$ and $R' = V(G) \setminus (L' \cup S')$, defines a valid x-y vertex-cut in $\hat{G}_{x,y}$. Moreover, if (x,y) is the distinguished pair and $L \subseteq A_{x,y}$ holds, then (L',S',R') must be a global minimum vertex-cut. We then return the value $c_{x,y} = w(\hat{S})$. The main challenge with this approach is to compute the desired set $A_{x,y}$ of vertices efficiently. We now describe our algorithm for doing so, for a fixed pair $(x,y) \in \Gamma$ of vertices. Throughout, we use a parameter $\tau' = \frac{\tau}{64|L|^2}$, and we denote by $\hat{S} \subseteq S$ the set containing all low-degree vertices $v \in S$ with $w(v) \geq \tau'$ (that is not known to the algorithm). We design two different algorithms for computing the set $A_{x,y}$ of vertices, that separately deal with the cases where $|\hat{S}|$ is large or small.

Case 1: $|\hat{S}| \geq 2^{12}|L|^3$. For every vertex $v \in V(G)$, we define a set $J(v) \subseteq V(G)$, that contains all low-degree vertices of $N^+_G(\beta(v))$ whose weight is at least τ'. Recall that we have established that, if $v \in L$, then $N^+_G(\beta(v)) \subseteq L \cup S$. It is then not hard to see that $J(v) \subseteq L \cup \hat{S}$, and, moreover, $J(v)$ must contain the vast majority of the vertices of $\hat{S}$. We then let $A_{x,y}$ contain all low-degree vertices $v \in V(G)$, for which $|J(v) \triangle J(x)| \leq 256|L|^3$ holds. We show that, if $x \in L$, then $L \subseteq A_{x,y}$ must hold. We provide an efficient randomized algorithm for constructing the set $A_{x,y}$ of vertices, that essentially samples $O(\log n)$ vertices from $J(x)$ uniformly at random, and, for each such sampled vertex v, adds to $A_{x,y}$ all low-degree vertices in the sets $\{\sigma(u) \mid u \in N^-_G(v) \setminus U\}$. In order to ensure that the algorithm is efficient and that the resulting set $A_{x,y}$ of vertices is small, we crucially rely on the assumption that no vertex of L is suspicious.

This provides an efficient algorithm for constructing the set $A_{x,y}$ of vertices with the desired properties in the case where $|\hat{S}| \geq 2^{12} \cdot |L|^3$. The main challenge in designing an algorithm for the case where $|\hat{S}|$ is small is that, if we use the same strategy as above, we can no longer guarantee that $|A_{x,y}|$ is small. Specifically, if $|\hat{S}|$ is sufficiently large (like in Case 1), then the vast majority of the vertices of $J(x)$ must lie in $\hat{S}$, and, from the definition of the set $A_{x,y}$ of vertices, for every vertex $v \in A_{x,y}$, some vertex in $J(v)$ must lie in $\hat{S}$ (and in fact a large fraction of the vertices of $\hat{S}$ must lie in $J(v)$). This allows us to bound the cardinality of $A_{x,y}$ by the total in-degree of all vertices in $\hat{S}$ times $O(n^\epsilon \cdot |L|)$ (the bound on $|\sigma(u)|$ for non-suspicious vertices u). Assume now that $|\hat{S}|$ is small, for example, that $|\hat{S}| \ll |L|$. In this case, the vast majority of the vertices of $J(x)$ may lie in L. In order to ensure that $L \subseteq A_{x,y}$, we need to allow $A_{x,y}$ to contain all vertices $v \in V(G)$ with $|J(v) \triangle J(x)| \leq 2|L|$, and in particular $A_{x,y}$ may contain vertices v with $J(v) \cap J(x) = \emptyset$. But then we can no longer guarantee that, for every vertex $v \in A_{x,y}$, $J(v) \cap \hat{S} \neq \emptyset$, and so we no longer have a way to bound $|A_{x,y}|$ appropriately. In view of these difficulties, we employ a completely different strategy that exploits the fact that $|\hat{S}|$ is small directly.

Case 2: $|\hat{S}| < 2^{12}|L|^3$. Our algorithm for this case is more involved. We use a parameter $\hat{\tau} = \frac{\tau}{8|L|}$, and slightly modify the definition of the sets $\beta(v)$ of vertices: for every vertex $v \in V(G)$, in order to construct the set $\beta(v)$, we perform the DFS search in graph G starting from v as before, except that now we only explore low-degree vertices of weight greater than $\hat{\tau}$ (instead of τ). For every vertex $u \in V(G)$, we define the vertex set $\sigma(u) = \{v \in V(G) \mid u \in \beta(v)\}$ exactly as before (except that we use the new definition of the vertex sets $\beta(v)$), and we define the set $U = \{u \in V(G) \mid |\sigma(u)| \geq 2n^\epsilon \cdot |L|\}$ of suspicious vertices exactly as before. The special case where some vertex in L is suspicious is also dealt with exactly as before, so we assume from now on that $U \cap L \neq \emptyset$.

We introduce a new notion of a *promising vertex set*, that is defined as follows. A set $Z \subseteq V(G)$ is a promising vertex set, if every vertex in Z is a low-degree vertex, and, additionally, every low-degree vertex $v \in S$ of weight

$w(v) \geq \hat{\tau}$ lies in Z.

Consider now some vertex $v \in L$. With the new modified definition of the vertex set $\beta(v)$, we are no longer guaranteed that $\beta(v) \subseteq L$. However, we show that $\beta(v)$ has another crucial property that our algorithm can exploit: if Z is any promising vertex set, then $\beta(v)$ must contain a low-degree non-suspicious vertex u' that is connected with an edge to some vertex of Z.

The key subroutine of our algorithm constructs a relatively small subset Z of vertices of G, such that, if (x, y) is the distinguished pair, then, with a sufficiently high probability, Z is a promising vertex set. The desired set $A_{x,y}$ of vertices is then obtained by including, for every vertex $u \in Z$, for every low-degree non-suspicious in-neighbor $u' \in N^-(u)$, all vertices of $\sigma(u')$ in $A_{x,y}$. From the above discussion, if Z is a promising vertex set, then $L \subseteq A_{x,y}$ must hold.

Lastly, we need to provide an algorithm for our key subroutine, that computes a relatively small set $Z \subseteq V(G)$ of vertices, such that, if (x, y) is the distinguished pair, then, with a sufficiently high probability, Z is a promising vertex set. In order to construct such a vertex set Z, we employ the local flow augmentation technique, that was first introduced in [5] and then further refined in [11, 6, 8].

In order to utilize the local flow augmentation technique, we use the "split graph" G' of the input graph G, that can be thought of as the edge-capacitated version of G: the set $V(G')$ of its vertices contains two copies of every vertex $v \in V(G)$, an *out-copy*, denoted by v^{out} and an *in-copy*, denoted by v^{in}. For every vertex $v \in V(G)$, graph G' contains an edge $e_v = (v^{\text{in}}, v^{\text{out}})$ of capacity $c(e_v) = w(v)$, that we refer to as the *special edge representing* v. Additionally, for every edge $e = (u, v) \in E(G)$, graph G' contains an edge $(u^{\text{out}}, v^{\text{in}})$ of capacity $W_{\max} = \Theta(n \cdot W)$, that we refer to as the *regular edge representing* e. Note that the value of the minimum x-y vertex-cut in G is equal to the value of the minimum x^{out}-y^{in} edge-cut in G', which, in turn, from the Max-Flow / Min-Cut theorem, is equal to the value of the maximum x^{out}-y^{in} flow in G'.

We use a graph $G'_{x,y}$, that is initially identical to G', and we denote x^{out} by s and y^{out} by t. Recall that, if $v \in V(G)$ is a high-degree vertex, then v may not lie in L. Therefore, for every high-degree vertex $v \in V(G)$, we can add a "shortcut" edge (v^{out}, t) of capacity ∞ to $G'_{x,y}$, without increasing the value of the maximum s-t flow in $G'_{x,y}$. We then let $G^h \subseteq G'_{x,y}$ be the graph that is obtained from $G'_{x,y}$ by deleting from it the copies of all vertices $v \in V(G)$ with $w(v) < \tau'$ (recall that $\tau' = \frac{\tau}{64|L|^2}$). Let $\mathsf{OPT}_{x,y}$ denote the value of the maximum s-t flow in graph G^h.

Our algorithm starts by computing an initial s-t flow f in G^h, whose value is close to $\mathsf{OPT}_{x,y}$: $\text{val}(f) \geq \mathsf{OPT}_{x,y} - 2|\hat{S}| \cdot \tau$, such that, for every vertex $v \in \hat{S}$, $f(v^{\text{in}}, v^{\text{out}}) = 0$ holds, via a simple procedure. Then we gradually augment the flow f via the local flow augmentation paradigm, over the course of $O(|L|^2 \cdot |\hat{S}|)$ iterations, where in every iteration, we send τ' flow units over a single augmenting flow-path in the residual flow network H of G^h. The main difference from the standard local flow augmentation algorithm is that we only employ augmenting paths of length at most $16|L|$, and we terminate the algorithm once the total number of vertices in $\{v^{\text{out}} \mid v \in V(G)\}$ reachable from s via paths of length at most $16|L|$ in the residual flow network H becomes sufficiently small. We then add to Z every low-degree vertex $v \in V(G)$, such that either (i) v^{out} is reachable from s via a path of length at most $16|L|$ in the residual flow network H obtained at the end of the local flow augmentation algorithm; or (ii) edge $(v^{\text{in}}, v^{\text{out}})$ lied on at least one of the augmenting paths employed by the local flow augmentation algorithm.

We prove that, if (x, y) is the distinguished pair, then, with a sufficiently high probability, the resulting set Z of vertices must be promising. The intuitive reason for this is as follows. Assume that (x, y) is the distinguished pair, and let $v \in S$ be a low-degree vertex with $w(v) \geq \hat{\tau}$. Then $v \in \hat{S}$ must hold, so, in the initial flow f, $f(v^{\text{in}}, v^{\text{out}}) = 0$ must hold. If the local flow augmentation algorithm ever sent flow on the edge $(v^{\text{in}}, v^{\text{out}})$, then v is added to Z, since then the edge $(v^{\text{in}}, v^{\text{out}})$ must have lied on one of the augmenting paths. Otherwise, at the end of the local flow augmentation algorithm, $f(v^{\text{in}}, v^{\text{out}}) = 0$ holds, and, from the properties of Maximum Flow, since $v \in S$, we must be able to send at least $w(v) \geq \hat{\tau}$ flow units from s to v^{in}, in the residual flow network $\tilde{H}$ of $G'_{x,y}$ with respect to the current flow f, such that this new flow only utilizes vertices of $G'_{x,y}$ that are out-copies of the vertices of L or in-copies of the vertices of $L \cup S$. It is then not hard to show that the residual flow network H of G^h with respect to the current flow f must contain an s-v^{in} path of length at most $16|L| - 1$, and so $v \in Z$

must hold. To summarize, in case where $|\hat{S}|$ is small, we use the local flow augmentation technique to compute a relatively small set Z of vertices of G, such that, if (x,y) is the distinguished pair, then, with a sufficiently high probability, Z is a promising vertex set. We then compute the vertex set $A_{x,y} \subseteq V(G)$ with the desired properties from Z as described above.

Weighted Dense Graphs We now provide a high-level overview of our $O\left(n^{2.677} \cdot \operatorname{poly}\log W\right)$-time algorithm for the weighted version of the problem, that is most useful in relatively dense graphs. We fix a distinguished cut (L,S,R) in the input vertex-weighted graph G exactly as before, and, as before, we assume that we are given an approximate estimate λ (to within factor 2) of $|L|$. As before, we can assume that $|L|$ is not too large, so, for example, $|L| \leq \sqrt{n}$, since otherwise we can use the algorithm for easy special cases, where the distinguished cut is balanced, that was described above.

As in our algorithm for the non-dense setting, we employ a simple randomized procedure to compute a collection Γ of roughly $\Theta\left(\frac{n}{|L|}\right)$ pairs of vertices of G, such that, with a sufficiently high probability, there exists some pair $(x^*,y^*) \in \Gamma$ (that we refer to as the *distinguished pair*) with $x^* \in L$ and $y^* \in R$. We can also ensure that, for every pair $(x,y) \in \Gamma$, $(x,y) \notin E(G)$. As before, the key part of our algorithm is a subroutine, that, given a pair $(x,y) \in \Gamma$, computes an x-y vertex-cut $(L_{x,y}, S_{x,y}, R_{x,y})$ in G, such that, if (x,y) is the distinguished pair, then, with a sufficiently high probability, $(L_{x,y}, S_{x,y}, R_{x,y})$ is a minimum x-y vertex-cut; we refer to this subroutine as the *main subroutine*. Our algorithm applies the main subroutine to every pair $(x,y) \in \Gamma$, and then outputs the smallest-value vertex-cut $(L_{x,y}, S_{x,y}, R_{x,y})$ from among the resulting cuts. In the remainder of this overview we focus on the description of the main subroutine.

Recall that $|\Gamma| \approx \Theta\left(\frac{n}{|L|}\right)$, and that our goal is to design an algorithm for global minimum vertex-cut with running time $O\left(n^{2.677} \cdot \operatorname{poly}\log W\right)$. Since the main subroutine is applied to every vertex pair in Γ separately, we need to ensure that its running time is bounded by $O\left(n^{1.677} \cdot |L| \cdot \operatorname{poly}\log W\right)$ — the running time that may be sublinear in m if $|L|$ is small. In particular, the main subroutine may not even be able to read the entire graph, and this is one of the main challenges in designing it.

At a high level, the main subroutine can be thought of as an adaptation of a similar subroutine of [8] to directed graphs (see Theorem 3.16 in [8]). While it is generally not difficult to extend their subroutine to the directed setting, our main contribution in the dense regime lies in significantly simplifying their algorithm and its analysis, while also obtaining a faster running time. Next, we provide a high-level overview of the procedure of [8] that is analogous to our main subroutine, and then describe our implementation of the main subroutine.

The Main Subroutine of [8]. We start by defining a new notion, that was implicitly used in [8] and in several previous works, that we call *shortcut flow*. Consider the input graph G and the given pair $(x,y) \in \Gamma$ of its vertices. We say that a collection E' of pairs of vertices of G is a collection of *shortcut edges* if, for every pair $(u,v) \in E'$, $v = y$ holds. We say that it is a *valid collection of shortcut edges*, if, for every pair $(u,y) \in E'$, $u \notin L$ (recall that (L,S,R) is the distinguished cut we fixed in G). Observe that, if $(x,y) \in \Gamma$ is the distinguished pair, and E' is a valid collection of shortcut edges, then (L,S,R) remains a valid x-y vertex-cut in the graph $G \cup E'$ that is obtained from G by inserting the edges of E'. An *x-y shortcut flow* is a pair (E',f), where E' is a set of shortcut edges, and f is an x-y vertex-capacitated flow in $G \cup E'$. We say that a shortcut flow (E',f) is *valid* if the set E' of shortcut edges is valid.

At a high level, the main subroutine of [8], given the input graph G and the pair $(x,y) \in \Gamma$ of its vertices, gradually constructs a shortcut x-y flow (E',f) in G, over the course of $O(\log(Wn))$ phases. Initially, $E' = \emptyset$ and $f = 0$. Then, in every phase, new edges may be added to E', and the flow f is augmented, so that its value becomes closer and closer to the value of the maximum x-y flow in G, which, in turn, is equal to the value of the minimum x-y vertex-cut. The subroutine guarantees that, if the vertex pair $(x,y) \in \Gamma$ is the distinguished pair, then, with high probability, E' remains a valid collection of shortcut edges throughout the algorithm, and, additionally, the gap $w(S) - \operatorname{val}(f)$ decreases by a constant factor from phase to phase. Therefore, after $O(\log(Wn))$ phases, $w(S) - \operatorname{val}(f) < \frac{1}{2}$ holds, and so the value $w(S)$ can be computed by rounding the value of the flow f up to the nearest integer.

While the above description conceptually matches the high-level structure of the subroutine of [8], it is imprecise

and omits important detail. Specifically, the algorithm of [8], in order to gradually augment the flow f, uses a standard paradigm of computing augmenting paths in the residual flow network, and then augmenting the flow f via these paths. This technique inherently only works with edge-capacitated, rather than vertex-capacitated graphs. In order to circumvent this difficulty, [8] do not directly use the shortcut flow as defined above, and instead use an analogous object defined with respect to the split graph G' of G. Each phase of the subroutine of [8] is then executed on the residual network G'_f of the split graph G' with respect to the current flow f, which leads to a rather complicated algorithm, due to the complex structure of the graph G'_f.

Our approach: well-behaving cuts. From the above description, the algorithm of [8] implementing the main subroutine can be thought of as gradually augmenting a shortcut x-y flow f from phase to phase, until its value becomes very close to the optimal one. In contrast, our algorithm uses a different object in order to make progress from phase to phase, that we refer to as a *well-behaving cut*. Consider an x-y vertex-cut (L', S', R') in G, and a parameter M that is an integral power of 2 (where possibly $M < 1$). We say that the cut (L', S', R') is *well-behaving for scale M*, if all of the following hold: (i) $\mathrm{Vol}^+(L') \leq \frac{n^2}{\gamma}$ (here, $\gamma = n^c$ for a constant $0 < c < 1$ is a parameter that we choose in order to optimize the algorithm's running time); (ii) $w(S') \leq w(S) + 10nM$; and (iii) the set $L' \cup S'$ contain all but at most $|L| \cdot \gamma$ vertices of $L \cup S$ whose weight is at least M. One can think of the scale parameter M as measuring the quality of the well-behaving cut. Our algorithm for the main subroutine starts with the x-y cut (L_0, S_0, R_0) in G where $L_0 = \{x\}$, $R_0 = \{y\}$, and $S_0 = V(G) \setminus \{x, y\}$; it is not hard to see that this cut must be well-behaving for a scale $M_0 = 10W$. It then performs $O(\log(Mn))$ phases, where, in the ith phase, it must produce an x-y vertex-cut (L_i, S_i, R_i) that is well-behaving for scale $M_i = \frac{M_0}{2^i}$. Let $z = O(\log(nW))$ be the smallest integer for which $M_z \leq \frac{1}{20n}$. We terminate the algorithm after z phases are completed, and output the vertex-cut (L_z, S_z, R_z) that was computed in Phase z. Since $w(S_z) - w(S) < 10nM_z < 1$ holds, and, since all vertex weights are integral, the cut (L_z, S_z, R_z) must be optimal.

We note that the algorithm of [8] produces an intermediate object in every phase, that they refer to as the "final cut". This object is an edge-cut in the residual flow network G'_f of the split-graph G' with respect to the current flow f, and it has some useful properties, whose description is somewhat complex and is omitted here. This cut is used by [8] in order to sparsify the graph G'_f, so that the flow augmentation can be computed efficiently. However, it is not hard to show that one can also derive a well-behaving x-y vertex-cut in G from their "final cut". Therefore, while the algorithm of [8] is not directly presented in this way, one can view it as following the high-level approach of computing better and better well-behaving cuts. The main drawback of their algorithm is that it explicitly focuses on maintaining an analogue of the shortcut x-y flow in the split-graph G' of G, that is gradually augmented over the course of the entire algorithm. As such, each phase of their algorithm needs to work with the corresponding residual flow network of G' with respect to this flow, whose structure is quite complex, adding an extra level of complexity to the algorithm and its analysis. In contrast, our algorithm focuses directly on obtaining better and better well-behaving cuts in the input graph G. While some steps involved in computing improved well-behaving cuts do require computing an x-y flow in the split graph G', this use of the split graph is limited to rather simple subroutines, and the resulting flow is discarded once the improved well-behaving cut is computed. Therefore, the majority of our algorithm works directly with the vertex-capacitated input graph G, which greatly simplifies the algorithm and its analysis. The main ingredient of our subroutine is then an algorithm for implementing a single phase, that we describe next.

Implementing a single phase. We now describe our algorithm for the ith phase, for $i \geq 1$. We assume that, at the beginning of the phase, we are given an x-y vertex-cut $(L_{i-1}, S_{i-1}, R_{i-1})$ in G. For brevity, we say that Condition (C) holds, if the vertex pair $(x, y) \in \Gamma$ is distinguished, and the cut $(L_{i-1}, S_{i-1}, R_{i-1})$ is well-behaving for scale M_{i-1}. Our goal is to produce an x-y vertex-cut (L_i, S_i, R_i) in G with the following property: if Condition (C) holds, then, with a sufficiently high probability, cut (L_i, S_i, R_i) must be well-behaving for scale $M_i = \frac{M_{i-1}}{2}$. The algorithm for the ith phase consists of three steps. In Step 1, we exploit the vertex-cut $(L_{i-1}, S_{i-1}, R_{i-1})$ in order to compute a valid shortcut x-y flow (E', f) in G, whose value is close to the value of the maximum x-y flow. In Step 2, the shortcut flow (E', f) is used in order to compute a set $A \subseteq V(G)$ of vertices with $\mathrm{Vol}^+_G(A) \leq \frac{n^2}{\gamma}$. We say that the vertex set A is *promising*, if there exists an x-y vertex-cut $(\hat{L}, \hat{S}, \hat{R})$ in G with $\hat{L} \subseteq A$, that is well-behaving for scale M_i; in fact, we require that this cut satisfies even slightly stronger properties. Our algorithm for Step 2 guarantees that, if Condition (C) holds, then the vertex set A that it computes is promising. We note

however that the algorithm for Step 2 does not compute the cut $(\hat{L}, \hat{S}, \hat{R})$ explicitly; instead it only explicitly computes the vertex set A and only guarantees the existence of the cut $(\hat{L}, \hat{S}, \hat{R})$ with the above properties, if Condition (C) holds. In the third and the last step, we exploit the vertex set A in order to compute the vertex-cut (L_i, S_i, R_i), such that, if Condition (C) holds, then cut (L_i, S_i, R_i) is well-behaving for scale M_i. Intuitively, the third step can be thought of as "approximately recovering" the vertex-cut $(\hat{L}, \hat{S}, \hat{R})$ that is guaranteed to exist by the definition of the promising set – since $(\hat{L}, \hat{S}, \hat{R})$ satisfies somewhat stronger properties than those required by a well-behaving cut, an "approximation" of $(\hat{L}, \hat{S}, \hat{R})$ must also be a well-behaving cut. We note that, at a very high level, our algorithm for recovering a well-behaving cut from a shortcut flow can be viewed as somewhat similar to the approach of [8], who obtain their "final cut" (an analogue of a well-behaving cut) via an intermediate object, that can be shown to implicitly encode a promising vertex set; this object is a subgraph of G'_f that they denote by J. However, both our algorithms for computing the promising vertex set from a shortcut flow, and for computing a well-behaving cut from a promising set, are faster and significantly simpler than the methods used by [8] for the analogous tasks.

Organization. We start with preliminaries in Section 2. We describe algorithms for several easy special cases in Section 3, and introduce some key tools that we use in our algorithms for both the unweighted and the weighted settings in Section 4. In Section 5, we present our algorithm for unweighted graphs, proving Theorem 1.2. We present our $O\left(mn^{11/12+o(1)} \cdot d^{1/12} \cdot \operatorname{poly}\log W\right)$-time algorithm for the weighted setting in Section 6, and our $O\left(n^{2.677} \cdot \operatorname{poly}\log W\right)$-time algorithm in Section 7, where we also complete the proof of Theorem 1.1 by combining these two algorithms.

2 Preliminaries

All logarithms in this paper are to the base of 2. We use the $\tilde{O}(\cdot)$ notation to hide $(\log n)^{O(1)}$ factors, where n is the number of vertices in the input graph. For an integer $k > 0$, we use $[k]$ to denote the set $\{1, \ldots, k\}$ of integers.

2.1 Graph-Theoretic Notation

Suppose we are given a directed graph $G = (V, E)$ with weights $w(v) \geq 0$ on its vertices $v \in V$. For a vertex $v \in V$, the set of its *out-neighbors* is $N^+_G(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$, and the set of its *in-neighbors* is $N^-_G(v) = \{u \in V(G) \mid (u, v) \in E(G)\}$. We denote by $N_G(v) = N^+_G(v) \cup N^-_G(v)$ the set of all neighbors of v. We also denote by $\deg^+_G(v) = |N^+_G(v)|$, $\deg^-_G(v) = |N^-_G(v)|$ and $\deg_G(v) = \deg^+_G(v) + \deg^-_G(v)$ the out-degree, the in-degree, and the total degree of v, respectively. Additionally, we denote by $\delta^-_G(v) = \{(u, v) \in E(G) \mid u \in V(G)\}$ and $\delta^+_G(v) = \{(v, u) \in E(G) \mid u \in V(G)\}$ the sets of all incoming and outgoing edges of v, respectively. Given a subset $X \subseteq V$ of vertices, its *volume* is $\operatorname{Vol}(X) = \sum_{v \in X} \deg_G(v)$, and its *weight* is $w(X) = \sum_{v \in X} w(v)$. Given a value $\tau \in \mathbb{R}$, we denote by $X^{\geq \tau} = \{v \in X \mid w(v) \geq \tau\}$ the set of all vertices of X whose weight is at least τ. We denote the *out-volume of* X by $\operatorname{Vol}^+_G(X) = \sum_{v \in X} \deg^+_G(v)$. We also denote by $N^+_G(X) = \left(\bigcup_{x \in X} N^+_G(x)\right) \setminus X$, and by $N^-_G(X) = \left(\bigcup_{x \in X} N^-_G(x)\right) \setminus X$ the sets of all out-neighbors and all in-neighbors of the set X, respectively, excluding the vertices that lie in X. Given two disjoint vertex sets $A, B \subseteq V(G)$, we denote by $E_G(A, B) = \{(u, v) \in E(G) \mid u \in A, v \in B\}$. Given a graph $G = (V, E)$ and a set $E' \subseteq V \times V$ of pairs of its vertices, we use $G \cup E'$ to denote the graph obtained by inserting the edges of $E' \setminus E(G)$ into G. We may omit the subscript G in the above notations when the graph G is clear from context.

For a path $P \subseteq G$, its *length* is $|E(P)|$ – the number of edges on P. For a pair $s, t \in V(G)$ of vertices, the *distance* from s to t, denoted $\mathsf{dist}_G(s, t)$, is the length of the shortest path connecting s to t; if no such path exists, then $\mathsf{dist}_G(s, t) = \infty$. Given a vertex $s \in V(G)$ and a subset $T \subseteq V(G)$ of vertices, the distance from s to T is $\mathsf{dist}_G(s, T) = \min_{t \in T}\{\mathsf{dist}_G(s, t)\}$. We may omit the subscript G in the above notations when it is clear from the context.

Adjacency-list representation of graphs. Some of our algorithms utilize adjacency-list representations of graphs, that we recap here. Suppose we are given a directed graph G with capacities $c(e)$ on its edges $e \in E(G)$ and weights $w(v)$ on its vertices $v \in V(G)$. The adjacency-list representation of G stores, for every vertex $v \in V(G)$, its weight $w(v)$, its in-degree $\deg^-_G(v)$, and its out-degree $\deg^+_G(v)$. Additionally, it stores a doubly-linked list $\mathsf{IN}(v)$, listing all edges $e \in \delta^-_G(v)$, together with the capacity $c(e)$ of each such edge. It also stores the edges of $\delta^-_G(v)$ in a separate efficient search structure, such as a binary search tree, that allows to look up an edge (v, u) by its endpoint u. Similarly, it stores a doubly-linked list $\mathsf{OUT}(v)$ of all edges in $\delta^+_G(v)$, together with the capacity $c(e)$ of each such edge. The edges of $\delta^+_G(v)$ are also stored in an efficient search structure, where an edge (u, v)

can be looked up by its endpoint u. If the edge capacities in G are undefined then we assume that they are unit, and similarly, if vertex weights in G are undefined then we assume that they are unit.

Vertex-Cuts in Directed Graphs and Global Minimum Vertex-Cut. Given a directed graph G with weights $w(v) \geq 0$ on its vertices $v \in V(G)$, a *vertex-cut* in G is a partition (L, S, R) of $V(G)$ with $L, R \neq \emptyset$, so that no edge of G connects a vertex of L to a vertex of R (but edges connecting vertices of R to vertices of L are allowed). The *value* of the cut is the total weight $w(S) = \sum_{v \in S} w(v)$ of the vertices of S. A *global minimum vertex-cut* is a vertex-cut (L, S, R) of minimum value. Given an input graph G with weights $w(v) \geq 0$ on its vertices, we denote by OPT the value of the global minimum vertex-cut in G.

Notice that we can assume without loss of generality that there is a global minimum vertex-cut (L, S, R) with $w(L) \leq w(R)$; if this is not the case, then we can consider the graph $\overline{G}$ that is obtained from G by reversing the direction of all its edges. Clearly, if (A, B, C) is a vertex-cut in G, then (C, B, A) is a vertex-cut in $\overline{G}$ of the same value, and vice versa. We can now focus on computing a minimum vertex-cut in $\overline{G}$. Therefore, from now on we assume that there is a global minimum vertex-cut (L, S, R) in G with $w(L) \leq w(R)$, and we will use the term "global minimum vertex-cut" to only include such cuts (in other words, a cut (L', S', R') with $w(L') > w(R')$ will not be considered a global minimum vertex-cut regardless of its value $w(S')$).

Consider now a directed vertex-weighted graph G 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 s-t vertex-cut*, if the value of the cut is minimized among all such cuts. Notice that, if there is an edge connecting s to t in G, then G does not contain a vertex-cut separating s from t; in such a case, we say that the value of the minimum s-t vertex-cut is infinite, and that the cut is undefined.

Induced Tripartitions and Vertex-Cuts. Let G be a directed graph, and let $A \subseteq V(G)$ be a non-empty subset of its vertices. Consider the tri-partition (L, S, R) of the vertices of G, where $L = A$, $S = N^+_G(A)$, and $R = V(G) \setminus (L \cup S)$. We say that (L, S, R) is the *tripartition of $V(G)$ induced by A*, or that A *induces* the tripartition (L, S, R). Note that, if $A \cup N^+_G(A) \neq V(G)$, then (L, S, R) is also valid vertex-cut in G. In such a case, we may say that (L, S, R) is the *vertex-cut induced by A*, and that A *induces* the vertex-cut (L, S, R). When the set A consists of a single vertex x, we may refer to $\big(x, N^+_G(x), V(G) \setminus \big(\{x\} \cup N^+_G(x)\big)\big)$ as the *tripartition induced by x*, and, if it is a valid vertex-cut, then we may refer to it as the *vertex-cut induced by x*. Note that if, for every vertex $v \in V(G)$, $N^+_G(v) = V(G) \setminus \{v\}$, then G does not have a valid vertex-cut. Otherwise, every vertex $v \in V(G)$ with $\deg^+_G(v) < |V(G)| - 1$ induces a valid vertex-cut in G.

Flows in vertex-weighted graphs. Suppose we are given a directed graph G with weights $w(v) \geq 0$ on its vertices $v \in V(G)$, and two special vertices s and t. An *s-t flow* in G is an assignment of a flow value $f(e) \geq 0$ to every edge $e \in E(G) \setminus (\delta^-(s) \cup \delta^+(t))$, such that, for every vertex $v \in V(G) \setminus \{s, t\}$, $\sum_{e \in \delta^+(v)} f(e) = \sum_{e \in \delta^-(v)} f(e)$ and $\sum_{e \in \delta^+(v)} f(e) \leq w(v)$ hold. The *value* of the flow is $\mathrm{val}(f) = \sum_{e \in \delta^+(s)} f(e)$. In our algorithms, we always assume that a flow f in a graph G is given by listing the collection $E^f \subseteq E(G)$ that contains all edges $e \in E(G)$ with $f(e) > 0$, together with the flow value $f(e)$ for each edge $e \in E^f$.

A *flow-path decomposition* of the s-t flow f consists of a collection $\mathcal{P}$ of s-t paths, and a value $f(P) > 0$ for every path $P \in \mathcal{P}$, such that $\sum_{P \in \mathcal{P}} f(P) = \mathrm{val}(P)$ and, for every edge $e \in E(G)$, $\sum_{\substack{P \in \mathcal{P}: \\ e \in E(P)}} f(P) \leq f(e)$. We sometimes refer to the paths in $\mathcal{P}$ as *flow-paths*. Recall that, from the Max-flow / Min-cut theorem, the value of the maximum s-t flow is equal to the value of the minimum s-t vertex-cut in any vertex-weighted graph.

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 $c(E_G(X, Y))$, that we also denote by $c_G(X, Y)$. We say that a cut (X, Y) is a *global minimum edge-cut for G*, if it is an edge-cut in G of minimum value. Given a pair s, t of vertices of G, we say that an edge-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 say that it is a *minimum s-t edge-cut* if it is an s-t edge-cut whose value is minimal among all s-t edge-cuts. 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 value $0 \leq f(e) \leq c(e)$ 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 $f(e) = 0$. 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 $\mathrm{val}(f) = \sum_{e \in \delta^+(s)} f(e)$. As in vertex-capacitated flows, we always assume that a flow f in a graph G is given by listing a collection $E^f \subseteq E(G)$ that contains all edges $e \in E(G)$ with $f(e) > 0$, together with the flow value $f(e)$ for each edge $e \in E^f$.

Residual Flow Network. Let $G = (V, E)$ be a directed graph with capacities $c(e) \geq 0$ on its edges $e \in E$, let $s, t \in V$ be a pair of vertices of G, and let f be an s-t flow in G. The residual flow network of G with respect to f, denoted G_f, is an edge-capacitated graph with $V(G_f) = V(G)$. The set $E(G_f)$ of edges is partitioned into two subsets: a set of *forward* edges and a set of *backward edges*. The set of forward edges of G_f contains, for every edge $(u, v) \in E(G)$ with $f(u, v) < c(u, v)$, the edge (u, v) of capacity $c_{G_f}(u, v) = c(u, v) - f(u, v)$. The set of backward edges of G_f contains, for every edge $(u, v) \in E(G)$ with $f(u, v) > 0$, the edge (v, u) of capacity $c_{G_f}(v, u) = f(u, v)$. For an edge $(u, v) \in G_f$, its capacity in G_f is called the *residual capacity of* (u, v). The following observation, whose proof is deferred to the full version of the paper, is a basic observation about the residual network G_f; similar observations were used in numerous previous works. For example, [3] proved a similar statement (see Theorem 10 of [3]), though their formulation is less general as it only covers the case where f is a maximum s-t flow.

OBSERVATION 2.1. *Let $G = (V, E)$ be a directed edge-capacitated graph, let $s, t \in V$ be a pair of vertices of G, and let f be an s-t flow in G. Consider the residual network $H = G_f$. For every edge $e \in E$, we denote its capacity in G by $c_G(e)$, and for every edge $e \in E(H)$, we denote its capacity in H by $c_H(e)$. Then, for every s-t edge-cut (X, Y) in G, $c_H(X, Y) = c_G(X, Y) - \mathrm{val}(f)$.*

2.2 Graph Derived via a Vertex Set One of the tools that we use in order to sparsify graphs is a graph that is derived from the input graph G via some vertex set $A \subseteq V(G)$, that we define next. We note that such graphs were used in past work, though to the best of our knowledge this notion was never formalized.

DEFINITION 2.2 (Graph derived via vertex set). *Let G be a directed graph with weights $w(v) \geq 0$ on its vertices $v \in V(G)$, and let $A \subseteq V(G)$ be a non-empty set of vertices. The* graph derived from G via A, *denoted by $G^{|A}$ is defined as follows. The vertex set of $G^{|A}$ is $A \cup N^+_G(A) \cup \{t\}$, where t is a new vertex that does not lie in G. The weight of the vertex t is set to $\max_{v \in V(G)} \{w(v)\}$, while the weights of all other vertices remain the same as in G. For simplicity, for every vertex $v \in V(G^{|A})$, its weight in $G^{|A}$ is still denoted by $w(v)$. The set of edges of $G^{|A}$ is defined as follows. First, we include, for every vertex $v \in A$, all edges of $\delta^+_G(A)$ in $G^{|A}$. Additionally, for every vertex $u \in N^+_G(A)$, we include the edge (u, t).*

In the following claim, whose proof is deferred to the full version of the paper, we summarize the central properties of the graph $G^{|A}$.

CLAIM 2.3. *Let G be a directed graph with weights $w(v) \geq 0$ on its vertices $v \in V(G)$, and let $A \subseteq V(G)$ be a non-empty set of vertices. Consider the graph $G^{|A}$ derived from G via A. Then $|E(G^{|A})| \leq 2\,\mathrm{Vol}^+_G(A)$ must hold, and graph $G^{|A}$ can be constructed in time $O(\mathrm{Vol}^+_G(A))$ given access to the adjacency-list representation of G and the vertex set A. Moreover, if (L, S, R) is a vertex-cut in G that is induced by the set L of vertices with $L \subseteq A$, then $(L, S, V(G^{|A}) \setminus (L \cup S))$ is a valid vertex-cut in $G^{|A}$.*

The following simple claim, whose proof is deferred to the full version of the paper, allows us to transform vertex-cuts in $G^{|A}$ to vertex-cuts in G.

CLAIM 2.4. *Let G be a directed graph with weights $w(v) \geq 0$ on its vertices $v \in V(G)$, and let $A \subseteq V(G)$ be a non-empty set of vertices with $A \cup N^+_G(A) \neq V(G)$. Consider the graph $G^{|A}$ derived from G via A, and let (L, S, R) be a vertex-cut in $G^{|A}$ with $t \in R$. Then $L \subseteq A$ must hold, and, moreover, $(L, S, V(G) \setminus (L \cup S))$ is a valid vertex-cut in G.*

We also use the following immediate corollary of Claim 2.4; the proof is deferred to the full version of the paper.

COROLLARY 2.5. *Let G be a directed graph with weights $w(v) \geq 0$ on its vertices $v \in V(G)$, and let $A \subseteq V(G)$ be a non-empty set of vertices with $A \cup N_G^+(A) \neq V(G)$. Let $x, y \in V(G)$ be a pair of vertices with $x \in A$ and $y \notin A \cup N_G^+(A)$. Consider the graph $G^{|A}$ derived from G via A, and let (L', S', R') be a minimum x-t vertex-cut in $G^{|A}$. Then $(L', S', V(G) \setminus (L' \cup S'))$ is a valid x-y vertex-cut in G. In particular, if we denote by $\mathsf{OPT}_{x,y}$ the value of the minimum x-y vertex-cut in G, and by $\mathsf{OPT}'_{x,t}$ the value of the minimum x-t vertex-cut in $G^{|A}$, then $\mathsf{OPT}'_{x,t} \geq \mathsf{OPT}_{x,y}$ must hold. Moreover, if there exists a global minimum vertex-cut (L, S, R) in G with $x \in L$, $y \in R$, and $L \subseteq A$, then $(L', S', V(G) \setminus (L' \cup S'))$ is a global minimum vertex-cut in G.*

2.3 An Assumption on Strictly Positive Vertex Weights Suppose we are given a simple directed 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 using a standard transformation as follows: for every vertex $v \in V(G)$, we let $w'(v) = n^2 \cdot w(v) + 1$. Let $\tilde{G}$ 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 $\tilde{G}$, then it must also be a global minimum vertex-cut in G. Therefore, in order to obtain an algorithm for the global minimum vertex-cut problem, it is enough to design an algorithm for the special case where all vertex weights are strictly positive.

Note that, if all vertex weights in G are strictly positive, and (L, S, R) is a global minimum vertex-cut in G, then every vertex $v \in S$ must have an in-neighbor $u \in N_G^-(v)$ that lies in L, since otherwise, by moving v from S to R, we obtain a vertex-cut of a smaller value. Therefore, $S = N_G^+(L)$ must hold, and in particular, (L, S, R) is a vertex-cut induced by L. Using a similar reasoning, every vertex $v \in S$ must have an out-neighbor $u' \in N_G^+(v)$ that lies in R, and so $S = N_G^-(R)$.

2.4 Parameters $W_{\max}$ and $W'_{\max}$ and The Split Graph In this subsection we assume that we are given a directed graph $G = (V, E)$ with integral weights $w(v) \geq 0$ on its vertices $v \in V(G)$.

Parameters $W_{\max}$ and $W'_{\max}$. These parameters are defined exactly like in [8]. We denote by $W'_{\max}(G)$ the smallest integral power of 2, such that $W'_{\max}(G) \geq 1 + \max_{v \in V} \{w(v)\}$ holds. We let $W_{\max}(G)$ be the smallest integral power of 2 with $W_{\max}(G) \geq 8n \cdot W'_{\max}(G)$. 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:

$$W_{\max}(G) \leq 16n \cdot W'_{\max}(G) \leq 32n \cdot (1 + \max_{v \in V} \{w(v)\}). \tag{2.1}$$

The Split Graph. The split graph is defined exactly like in [8], except that we extend the definition to directed graphs. Given a directed vertex-weighted graph G, its *split graph* is a directed graph G', that 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 $V(G') = \left\{v^{\text{in}}, v^{\text{out}} \mid v \in V\right\}$. The set of edges of G' is partitioned into two subsets: the set E^{spec} of *special* 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 $e_v = (v^{\text{in}}, v^{\text{out}})$, 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 a regular edge $(x^{\text{out}}, y^{\text{in}})$ of capacity $c(x^{\text{out}}, y^{\text{in}}) = W_{\max}(G)$ to G', representing e. This completes the definition of the split graph G'. 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 $O(|E(G')|)$.

Given a subset $Z \subseteq V(G)$ of vertices of G, we denote by $Z^{\text{in}} = \left\{v^{\text{in}} \mid v \in Z\right\}$, $Z^{\text{out}} = \{v^{\text{out}} \mid v \in Z\}$, and $Z^* = Z^{\text{in}} \cup Z^{\text{out}}$. We also denote by $V^{\text{in}} = \left\{v^{\text{in}} \mid v \in V(G)\right\}$ and $V^{\text{out}} = \{v^{\text{out}} \mid v \in V(G)\}$.

Note that, for every pair $s, t \in V(G)$ of vertices, the value of the 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 $s, t \in G$.

The following two simple observations, that were proved implicitly or explicitly numerous times in prior work, show that an s-t vertex-cut in G can be transformed into an s^{out}-t^{in} edge-cut of the same value in the split graph, and vise versa. The proofs of the observations are deferred to the full version of the paper.

OBSERVATION 2.6. *Let G be a directed graph with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V(G)$, let $s, t \in V(G)$ be a pair of distinct vertices, and let (L, S, R) be an s-t vertex cut in G. Consider the corresponding split graph G', and for every edge $e \in E(G')$, denote its capacity in G' by $c(e)$. Let $X = L^* \cup S^{\text{in}}$ and $Y = V(G') \setminus X = S^{\text{out}} \cup R^*$. Then (X, Y) is an s^{out}-t^{in} edge-cut in G' whose value is equal to $w(S)$.*

OBSERVATION 2.7. *Let G be an directed 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. Consider the corresponding split graph G', and for every edge $e \in E(G')$, denote its capacity in G' by $c(e)$. Let (X, Y) be an s^{out}-t^{in} edge-cut in G' whose value $c(X, Y) < W_{\max}(G)$. Let $L = \{v \in V(G) \setminus \{t\} \mid v^{\text{out}} \in X\}$, $S = \left\{v \in V(G) \mid v^{\text{in}} \in X, v^{\text{out}} \in Y\right\}$, and let $R = V(G) \setminus (L \cup S)$. Then (L, S, R) is an s-t vertex-cut in G, and $w(S) \leq c(X, Y)$. Moreover, if we let (L, S', R') be the tripartition of $V(G)$ induced by L, then (L, S', R') is an s-t vertex-cut in G and $S' \subseteq S$ must hold; in particular, $w(S') \leq c(X, Y)$.*

We obtain the following simple corollary of Observation 2.7, whose proof follows standard arguments and is deferred to the full version of the paper.

COROLLARY 2.8. *Let G be an directed 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 such that $(s, t) \notin E(G)$. Consider the corresponding split graph G'. Then, the value of the minimum s-t vertex-cut in G is equal to the value of the minimum s^{out}-t^{in} edge-cut in G', and this value is less than $\frac{W_{\max}(G)}{2}$.*

A split graph with shortcuts. Assume that we are given a directed graph $G = (V, E)$ with integral weights $w(v) \geq 1$ on its vertices $v \in V(G)$, and consider the corresponding split graph G'. We will sometimes use a *split graph with shortcuts*, that is defined as follows. Let $B \subseteq V(G')$ be any subset of vertices of G'. The corresponding split graph with B-shortcuts, denoted by $G''(B)$, is obtained from the graph G' by first adding a new vertex t to it, and then connecting every vertex $v \in B$ to the vertex t with a directed edge whose capacity is $W_{\max}$. These newly added edges are considered regular edges of the resulting graph $G''(B)$.

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 [7, 36]. The algorithm of [36] 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 [33, 19] in order to compute the optimal flow. In particular, the flow that they obtain is integral, provided all edge capacities are integral as well (see Lemma 4.1 in [36]). This result is summarized in the following theorem (see Theorem 1.1 and Lemma 4.1 in [36]).

THEOREM 2.9. *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 $O\left(m^{1+o(1)} \cdot \log U \cdot \log C\right)$.*

We also use the following immediate corollary of Theorem 2.9, whose proof follows standard arguments and can be found, e.g. in [8].

COROLLARY 2.10 (Corollary 2.9 in [8]). *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 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 $O\left(m^{1+o(1)} \cdot \log U\right)$.*

Finally, we use the following immediate corollary of Corollary 2.10, whose proof also uses standard techniques. The corollary was proven for undirected graphs in [8] (see Corollary 2.10). The proof for directed graphs is esentially identical, and is deferred to the full version of the paper.

COROLLARY 2.11. *There is a deterministic algorithm, that receives as input a directed 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, such that no edge of G connects a vertex of S to a vertex of T. 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 A Subgraph Oracle Our algorithm uses a subgraph oracle, that was defined in [8] for undirected graphs; we extend their definition to directed graphs in a straightforward manner.

DEFINITION 2.12 (A Subgraph Oracle). *A (δ,τ)-subgraph oracle, for parameters $0<\delta\leq 3/4$ and $\tau\geq 2$, receives as input a directed n-vertex graph G in the adjacency list representation and a set $Z\subseteq V(G)$ of its vertices. The oracle must return a partition (Y^h,Y^ℓ) of $V(G)$, and a collection E' of at most $n^{2-\delta}\cdot\log^2\tau$ edges of G, such that $E'=E_G(Y^\ell,Z)$ holds. We say that the oracle* errs, *if, for some vertex $v\in Y^h$, $|N^+_G(v)\cap Z|<\frac{n^{1-\delta}}{1000\log n}$ holds, or, for some vertex $v'\in Y^\ell$, $|N^+_G(v)\cap Z|>n^{1-\delta}\cdot\log\tau$ holds. We require that the probability that the oracle errs is at most $\frac{1}{n^5\cdot\log^4\tau}$.*

The following lemma is a straightforward extension of an analogous result of [8] to directed graphs, that provides a relatively efficient algorithm for responding to up to n subgraph oracle queries in bulk. We also slightly adapt the statement by explicitly stating the dependence of the running time on the number of queries, which is straightforward to do. The lemma statement uses the matrix multiplication exponent ω, whose current bound is $\omega\leq 2.371552$ [37]. The proof is essentially identical to that from [8] and is deferred to the full version of the paper.

LEMMA 2.13 (Extension of Lemma 3.17 in [8] to directed graphs). *There is a deterministic algorithm, that is given as input a directed n-vertex graph G, parameters $\frac{1}{\sqrt{\log n}}<\delta\leq\frac{3}{4}$ and $\tau>0$ that is greater than a large enough constant, and $q=n^{1-\epsilon}$ queries $(Z_1,\ldots,Z_q)$ to the (δ,τ)-subgraph oracle on G, for any $0\leq\epsilon\leq 1$. 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 $\frac{1}{n^9\cdot\log^4\tau}$. The running time of the algorithm is:*

$O\left(\left(n^{3-(\delta+\epsilon)\cdot 2(3-\omega)/(5-\omega)+o(1)}+n^{2+\delta+o(1)}\right)\cdot\operatorname{poly}\log\tau\right)\leq O\left(\left(n^{3-0.478(\delta+\epsilon)+o(1)}+n^{2+\delta+o(1)}\right)\cdot\operatorname{poly}\log\tau\right)$, *where ω is the matrix multiplication exponent.*

3 Algorithms for Easy Special Cases Suppose we are given a directed graph G with integral weights $w(v)\geq 1$ on its vertices. Observe first that, if G is not a strongly connected graph, then there is a partition (L,R) of vertices of $V(G)$ into two non-empty subsets, so that no edge connects a vertex of L to a vertex of R, and so the value of the global minimum vertex-cut in G is 0. Moreover, by performing two DFS searches from an arbitrary vertex v of G, one in the graph G itself, and one in the graph obtained from G by reversing the direction of all its edges, we can check, in time $O(m)$ whether G is strongly connected, and, if this is not the case, compute the vertex-cut of value 0. Therefore, we will assume from now on that the input graph G is strongly connected.

In the following theorem we provide a simple algorithm for some additional easy special cases of the problem: namely, where there is a global minimum cut (L,S,R), such that either $|L|$ is sufficiently large, or $\operatorname{Vol}_G(L)$ is high. The algorithm closely resembles algorithms that were employed in previous work for similar settings. The proof of the following theorem is deferred to the full version of the paper.

THEOREM 3.1. *There is a randomized algorithm, that we denote by* AlgSpecial, *that receives as input a directed n-vertex and m-edge graph $G=(V,E)$ with integral weights $1\leq w(v)\leq W$ on its vertices $v\in V$, such that G contains some vertex-cut, together with a parameter $0<\epsilon<1$. Let $d=\frac{2m}{n}$ denote the average vertex degree in G. The algorithm returns a vertex-cut (L',S',R') in G. Moreover, if there exists a global minimum vertex-cut (L,S,R) in G, for which either $|L|\geq n^\epsilon$ or $\operatorname{Vol}_G(L)\geq n^\epsilon\cdot d$ hold, then, with probability at least $(1-1/n^2)$, the vertex-cut (L',S',R') is a global minimum vertex-cut. The running time of the algorithm is $O(mn^{1-\epsilon+o(1)}\log W)$.*

4 Some Useful Tools In this section we introduce several additional tools that will be used in our algorithms for both unweighted and weighted graphs.

Suppose we are given a directed n-vertex and m-edge graph $G=(V,E)$ with integral weights $1\leq w(v)\leq W$ on its vertices $v\in V$ and a parameter $0<\epsilon<1$. Throughout, we denote by $d=2m/n$ the average vertex degree in G. Recall that algorithm AlgSpecial from Theorem 3.1 allows us to compute a global minimum vertex-cut in G in the special case where there exists a global minimum vertex-cut (L,S,R) in G, for which either $|L|\geq n^\epsilon$ or $\operatorname{Vol}_G(L)\geq n^\epsilon\cdot d$ hold. Therefore, for some of our algorithms, it is sufficient to consider the case where, for every global minimum vertex-cut (L,S,R) in G, $|L|<n^\epsilon$ and $\operatorname{Vol}_G(L)<n^\epsilon\cdot d$. Note that, if (L,S,R) is a global minimum vertex-cut in G, then, for every vertex $v\in S$, there must be an edge (u,v) with $u\in L$, since otherwise

we could move v from S to R and obtain a valid vertex-cut of a smaller value. Therefore, we can assume that $|S| \leq \mathrm{Vol}(L) \leq n^{\epsilon} \cdot d$ holds as well. To summarize, some of our algorithms will assume that there is a global minimum vertex-cut (L, S, R) in G with the following properties:

P1. $|L| < n^{\epsilon}$;

P2. $|S| < n^{\epsilon} \cdot d$; and

P3. $\mathrm{Vol}_G(L) < n^{\epsilon} \cdot d$.

This motivates our definition of high-degree vertices:

DEFINITION 4.1 (High-degree vertices). *We say that a vertex $v \in V(G)$ is a* high-degree vertex *if $\deg_G(v) \geq d \cdot n^{\epsilon}$, where d is the average vertex degree in G; otherwise, it is a* low-degree vertex. *We denote by $V^{\mathrm{hd}} = \{v \in V(G) \mid \deg_G(v) \geq d \cdot n^{\epsilon}\}$ the set of all high-degree vertices of G, and by $V^{\mathrm{ld}} = \{v \in V(G) \mid \deg_G(v) < d \cdot n^{\epsilon}\}$ the set of all low-degree vertices.*

Note that, if (L, S, R) is a global minimum vertex-cut for which Property P3 holds, then set L may not contain high-degree vertices.

4.1 Selection of Terminals and Parameter λ It will be convenient for us to assume that the algorithm is given a correct (approximate) estimate λ on the value $|L|$. Additionally, in some of our algorithms, we need to select a relatively small collection T of vertices of G called terminals, so that at least one vertex v of T lies in L, and the degree of v in G is not much higher than the average degree of all vertices in L. In the following claim, whose proof is deferred to the full version of the paper, we provide a simple algorithm that allows us to achieve both these goals.

CLAIM 4.2. *There is a randomized algorithm, that we call* AlgSelectTerminals*, that receives as input an n-vertex directed graph G with integral weights $w(v) \geq 1$ on its vertices $v \in V(G)$, and returns a value $1 \leq \lambda \leq n$ that is an integral power of 2, together with a set T of at most $\left\lceil \frac{100n \log n}{\lambda} \right\rceil$ vertices of G. We say that the algorithm is* successful *with respect to a fixed global minimum vertex-cut (L, S, R), if (i) $\frac{\lambda}{2} \leq |L| \leq \lambda$; (ii) $T \cap L \neq \emptyset$; and (iii) there is a vertex $v \in T \cap L$ with $\deg_G(v) \leq 4 \left\lceil \frac{\mathrm{Vol}_G(L)}{\lambda} \right\rceil$. Let (L, S, R) be a global minimum vertex-cut in G. The running time of the algorithm is $O(n)$, and it is successful with respect to (L, S, R) with probability at least $\frac{1}{4 \log n}$.*

4.2 A New Tool: an Approximate Furthest Min-Cut It is well known that, for any edge-capacitated graph G and a pair $x, y \in V(G)$ of distinct vertices, there exists a minimum x-y edge-cut (X, Y) in G with the following property: for every other minimum x-y edge-cut (X', Y') in G, $X' \subseteq X$ holds. Such a cut (X, Y) is sometimes referred to as the *furthest minimum x-y cut* (in fact, an analogous notion of "closest min-cut" was also used in the past, perhaps even more often, see e.g. [2]). The furthest minimum x-y cut (X, Y) can be computed by first computing a maximum x-y flow f in G, and then letting X contain all vertices $v \in G$ such that there is no v-y path in the resulting residual flow network G_f. In this section, we generalize the notion of furthest cuts to capture not just exact minimum x-y cuts, but also approximate ones, and provide an efficient algorithm for computing such cuts. We refer to the corresponding problem as the *Approximate Furthest Min-Cut* problem, or AFMC for short. We distinguish between the edge- and the vertex-versions of the problem, that we refer to as edge-AFMC and vertex-AFMC, respectively. We start by defining the edge version of the problem.

DEFINITION 4.3 (Edge-AFMC). *In the edge version of the Approximate Furthest Min-Cut problem (edge-AFMC), the input consists of a directed n-vertex and m-edge graph G with integral capacities $0 \leq c(e) \leq W$ on its edges $e \in E(G)$, a pair $x, y \in V(G)$ of distinct vertices, and an integral parameter $\alpha > 0$. We denote the value of the minimum x-y edge-cut in G by $\mathsf{OPT}^E_{x,y}$. The goal is to compute an x-y edge-cut (X, Y) in G of value $c(X, Y) \leq \mathsf{OPT}^E_{x,y} + \alpha$, such that, for every other x-y edge-cut (X', Y') the following holds:*

$$|X' \setminus X| \leq \left(c(X', Y') - \mathsf{OPT}^E_{x,y}\right) \cdot \frac{n}{\alpha}. \tag{4.1}$$

Observe that, unlike in the case of the exact furthest minimum edge-cut, we no longer require that the cut (X, Y) is the minimum x-y edge-cut; instead, it is sufficient that its value is within an additive α value from the optimal one. Additionally, for all $\beta > 0$, for every other x-y edge-cut (X', Y') whose value is at most $\mathsf{OPT}^E_{x,y} + \beta$, we only require that all but at most $\frac{\beta \cdot n}{\alpha}$ vertices of X' are contained in X, relaxing the stricter requirement that $X' \subseteq X$ in the exact furthest minimum edge-cut. In the following theorem, whose proof is deferred to the full version of the paper, we provide an efficient algorithm for the edge-AFMC problem.

THEOREM 4.4. *There is a deterministic algorithm for the edge-*AFMC *problem with running time* $O\left((n+m)^{1+o(1)} \log W \cdot \log(2+\alpha)\right)$.

In our algorithm for the weighted version of global minimum vertex-cut in dense graphs, we will need to solve the vertex version of the AFMC problem, that we refer to as *vertex-*AFMC and define next. The vertex version is very similar to the edge version, except that it naturally uses vertex-cuts (L, S, R) and (L', S', R') instead of the edge-cuts (X, Y) and (X', Y'), and it requires that the cardinality of $(L' \cup S') \setminus (L \cup S)$ is bounded, instead of the cardinality of $X' \setminus X$.

DEFINITION 4.5 (Vertex-AFMC). *In the vertex version of the Approximate Furthest Min-Cut problem (vertex-*AFMC*), the input is a directed n-vertex and m-edge graph G with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V(G)$, a pair of distinct vertices $x, y \in V(G)$ such that $(x, y) \notin E(G)$, and an integral parameter $0 < \alpha < \frac{W \max(G)}{2}$. We denote the value of the minimum x-y vertex-cut in G by $\mathsf{OPT}^V_{x,y}$. The goal is to compute an x-y vertex-cut (L, S, R) in G, with $w(S) \leq \mathsf{OPT}^V_{x,y} + 2\alpha$, such that, for every other x-y vertex-cut (L', S', R'), the following holds:*

$$|(L' \cup S') \setminus (L \cup S)| \leq \left(w(S') - \mathsf{OPT}^V_{x,y}\right) \cdot \frac{n}{\alpha}. \tag{4.2}$$

We note that Condition 4.2 in the above definition is an analogue of Condition 4.1 for edge-AFMC. In particular, it requires that, for every value $\beta \geq 0$, for every x-y vertex-cut (L', S', R') of value $w(S') \leq \mathsf{OPT}^V_{x,y} + \beta$, all but at most $\frac{\beta \cdot n}{\alpha}$ vertices of $L' \cup S'$ are contained in $L \cup S$.

In the following theorem, whose proof is deferred to the full version of the paper, we provide an efficient algorithm for the vertex-AFMC problem, by reducing it to edge-AFMC via a split graph, and then employing the algorithm from Theorem 4.4.

THEOREM 4.6. *There is a deterministic algorithm for the vertex-*AFMC *problem with running time* $O\left((n+m)^{1+o(1)} \log W \cdot \log(2+\alpha)\right)$.

4.3 Maximum s-t Flow with Small Support For some of our algorithms, we need to be able to quickly compute a maximum s-t flow f in a vertex-weighted directed n-vertex graph, such that the flow f is non-zero on at most $O(n)$ edges. In the following theorem we provide an algorithm for computing such a flow f in time that is almost-linear in the number of edges of the input graph. We note that the existence of such a flow was already shown in [34] (see Lemma 2.5 in [34]), and their proof also implicitly provides an algorithm for computing it. Unfortunately, the running time of this algorithm is at least $\Omega(nm)$ in the worst case, which is too slow for our purposes. We build on the techniques from [34] to provide an algorithm with an almost linear runing time. The algorithm is summarized in the following theorem, whose proof is deferred to the full version of the paper.

THEOREM 4.7. *There is a randomized algorithm, whose input consists of n-vertex and m-edge graph G with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V(G)$ and a pair $s, t \in V(G)$ of its vertices. The algorithm computes a maximum s-t flow f in G that obeys the vertex capacities (defined by their weights), such that f is integral, and, moreover, the total number of edges $e \in E(G)$ with $f(e) > 0$ is at most $4n$. The expected running time of the algorithm is $O(m^{1+o(1)} \cdot \log W)$.*

5 Algorithm for Unweighted Directed Global Minimum Vertex-Cut In this section we design an algorithm for the unweighted version of Global Minimum Vertex-Cut, proving Theorem 1.2. We use the following theorem, that provides an $O(n^{2+o(1)})$-time algorithm for the problem, and was proved in [25] (their initial theorem statement was conditional on the existence of a near-linear time algorithm for maximum s-t flow, but since such an algorithm (albeit with an almost-linear running time) was shown by [7, 36], it now holds unconditionally).

THEOREM 5.1 (follows from Theorem 1.2 in [25] and [7, 36]). *There is a randomized algorithm that, given a simple directed unweighted n-vertex and m-edge graph G that contains some vertex-cut, returns a vertex-cut (L', S', R') in G, so that, with probability at least 0.99, (L', S', R') is a global minimum vertex-cut. The running time of the algorithm is $O\left(n^{2+o(1)}\right)$.*

The main technical result of this section is the following theorem, that provides an algorithm for the unweighted directed global minimum vertex-cut problem with running time $O(m^{1+o(1)} \cdot \hat{k})$, where $\hat{k}$ is a "guess" on the value of the optimal solution that the algorithm receives as input.

THEOREM 5.2. *There is a randomized algorithm, that, given a simple directed unweighted n-vertex and m-edge graph G that contains some vertex-cut, together with an integer $1 \leq \hat{k} \leq \frac{n}{1000}$, computes a vertex-cut (L', S', R') in G, in time $O\left(m^{1+o(1)} \cdot \hat{k}\right)$. Moreover, if we denote by k the value of the global minimum vertex-cut in G, and if $\frac{\hat{k}}{2} < k \leq \hat{k}$ holds, then, with probability at least $\left(1 - \frac{1}{n}\right)$, (L', S', R') is a global minimum vertex-cut in G.*

We prove Theorem 5.2 in the remainder of this section, after completing the proof of Theorem 1.2 using it. We assume that $m \geq n/2$ holds, since otherwise the input graph G is not strongly connected, and a vertex-cut of value 0 can be computed in time $O(m)$ by performing two DFS searches. Let $z = \lfloor \log(n/1000) \rfloor$. For all $0 \leq i \leq z$, we execute the algorithm from Theorem 5.2 with the parameter $\hat{k} = 2^i$; we denote this algorithm by $\mathcal{A}_i$. We also denote by $\mathcal{A}_{z+1}$ the algorithm from Theorem 5.1. We execute all algorithms $\mathcal{A}_0, \ldots, \mathcal{A}_z, \mathcal{A}_{z+1}$ in parallel. For all $0 \leq i \leq z+1$, we may execute Algorithm $\mathcal{A}_i$ in its entirety (in which case we say that the algorithm has run its course), or we may choose to terminate it prematurely.

Whenever, for any $0 \leq i \leq z$ algorithm $\mathcal{A}_i$ terminates after running its course, if the cut (L_i, S_i, R_i) that it returns has $|S_i| \leq 2^i$, then we immediately terminate algorithms $\mathcal{A}_{i+1}, \ldots, \mathcal{A}_z$ prematurely, and if, additionally, $2^i \cdot m < n^2$, then we also terminate Algorithm $\mathcal{A}_{z+1}$ prematurely. We then record the vertex-cut (L_i, S_i, R_i), and refer to it as a *candidate solution*. If Algorithm $\mathcal{A}_{z+1}$ terminates after running its course, then we record the vertex-cut $(L_{z+1}, S_{z+1}, R_{z+1})$ as a candidate solution, and we terminate prematurely all algorithms $\mathcal{A}_0, \ldots, \mathcal{A}_z$ that have not terminated yet.

Once every algorithm $\mathcal{A}_j$ terminates (either because it has run its their course, or because we terminated it prematurely), we output the smallest-value candidate solution. This completes the description of the algorithm. We now analyze its running time and correctness.

Observe first that the running time of the algorithm may not exceed $O\left(n^{2+o(1)}\right)$. Indeed, if Algorithm $\mathcal{A}_{z+1}$ is allowed to run its course, then it terminates in time $O\left(n^{2+o(1)}\right)$, and, once it does so, all other algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_z$ are terminated. Otherwise, Algorithm $\mathcal{A}_{z+1}$ was terminated prematurely. This may only happen if, for some $0 \leq i \leq z$ with $2^i \cdot m < n^2$, Algorithm $\mathcal{A}_i$ terminates with a cut (L_i, S_i, R_i), and $|S_i| \leq 2^i$ holds. Recall that, from Theorem 5.2, for all $0 \leq i' \leq i$, the running time of algorithm $\mathcal{A}_{i'}$ is bounded by $O\left(m^{1+o(1)} \cdot 2^{i'}\right) \leq O\left(m^{1+o(1)} \cdot 2^i\right) \leq O\left(n^{2+o(1)}\right)$. Once Algorithm $\mathcal{A}_i$ terminates, all algorithms $\mathcal{A}_{i+1} \ldots, \mathcal{A}_{z+1}$ are terminated as well, and so the total running time of the entire algorithm is bounded by $O\left(n^{2+o(1)}\right)$.

Let k denote the value of the global minimum vertex-cut in G, and let $i \geq 0$ be the unique integer for which $2^{i-1} < k \leq 2^i$ holds. Assume first that $i > z$, so $k \geq \frac{n}{2000}$. In this case, $m \cdot k \geq \Omega(n^2)$, and, from the above discussion, the running time of the algorithm is $O\left(n^{2+o(1)}\right) \leq O\left(\min\left\{n^{2+o(1)}, m^{1+o(1)} \cdot k\right\}\right)$, as required. Moreover, for all $0 \leq i \leq z$, algorithm $\mathcal{A}_i$ may not terminate with a vertex-cut (L_i, S_i, R_i) of value $|S_i| \leq 2^i < k$, so Algorithm $\mathcal{A}_{z+1}$ was allowed to run its course, and, from Theorem 5.1, with probability at least 0.99, the cut $(L_{z+1}, S_{z+1}, R_{z+1})$ that it returns (which must be the only candidate solution for our algorithm) is optimal.

Therefore, we assume from now on that $i \leq z$. Recall that the running time of Algorithm $\mathcal{A}_i$ is bounded by $O\left(m^{1+o(1)} \cdot 2^i\right) \leq O\left(m^{1+o(1)} \cdot k\right)$. From the choice of the value of i, for $0 \leq i' < i$, algorithm $\mathcal{A}_{i'}$ may not terminate with a cut of value at most $2^{i'} < k$. Therefore, one of the algorithms $\mathcal{A}_i$ or $\mathcal{A}_{z+1}$ must have run its course. If it is the former, then, with probability at least $\left(1 - \frac{1}{n}\right)$, the cut (L_i, S_i, R_i) that $\mathcal{A}_i$ outputs has $|S_i| = k$. If this is indeed the case, then the entire algorithm terminates, and its running time is bounded by $O\left(m^{1+o(1)} \cdot k\right)$. With probability at most $\frac{1}{n}$, the cut (L_i, S_i, R_i) that Algorithm $\mathcal{A}_i$ returns is not optimal, and then the running

time of the entire algorithm is bounded by $O\left(n^{2+o(1)}\right)$. It is now easy to verify that the expected running time of the entire algorithm is bounded by $O\left(\min\left\{n^{2+o(1)}, m^{1+o(1)}\cdot k\right\}\right)$, and, with probability at least $\frac{1}{2}$, the cut that it returns is a global minimum vertex-cut. In order to complete the proof of Theorem 1.2, it is now enough to prove Theorem 5.2, which we do in the remainder of this section.

Proof of Theorem 5.2 We assume that the the number of vertices n in the input graph is greater than a sufficiently large constant, so that, for example, $n > \log^{20} n$ holds; if this is not the case, then we can use brute-force search to find and return a global minimum vertex-cut of G, in time that is upper-bounded by a sufficiently large constant.

Distinguished cut. Throughout the algorithm, for the sake of the analysis, we fix a global minimum vertex-cut (L,S,R) in G, that minimizes $|L|$, that we refer to as the *distinguished cut*. As before, we assume without loss of generality that $|L|\leq|R|$, since otherwise we can equivalently solve the problem on the graph $\overline{G}$ that is obtained from G by reversing the direction of its edges. Notice that, from the selection of the distinguished cut, it must be induced by the vertex set L.

Fixing an arbitrary vertex-cut. Our algorithm may sometimes need to return an arbitrary vertex-cut in G. For convenience, we compute such a vertex-cut $(\tilde{L},\tilde{S},\tilde{R})$ at the beginning of the algorithm, as follows. Recall that the input graph G has some vertex-cut, so for some vertex $v\in V(G)$, $\deg^+_G(v)<n-1$ must hold. We let v be any such vertex, and we let $(\tilde{L},\tilde{S},\tilde{R})$ be the tripartition of $V(G)$ induced by v; clearly, $(\tilde{L},\tilde{S},\tilde{R})$ is a valid vertex-cut in G. Note that the cut $(\tilde{L},\tilde{S},\tilde{R})$ can be computed in time $O(m)$, and we do so at the beginning of the algorithm. Generally, whenever our algorithm needs to output an arbitrary vertex-cut in G (which, intuitively, means that it failed), we output this fixed cut $(\tilde{L},\tilde{S},\tilde{R})$.

High-level overview of the algorithm. Our algorithm starts with a preprocessing step, in which we compute two disjoint sets T, T' of vertices whose cardinalities are bounded by roughly $\tilde{O}\left(\frac{m}{\mathrm{Vol}^+(L)}\right)$, and another vertex $t^*\in V(G)$ using a simple randomized procedure that ensures that, with a sufficiently high probability, $T\cap L\neq\emptyset$, $T'\cap L=\emptyset$ and $t^*\in R$ holds. We say that the good event $\mathcal{E}^*$ happens if these conditions are indeed satisfied. Throughout the algorithm, we refer to the vertices in the sets T and T' as *terminals* and *anti-terminals*, respectively. We then compute a hierachical partition of the collection T of terminals. The hierarchical partition has $z=O(\log|T|)$ levels. For all $0\leq i\leq z$, the *level-i partition*, denoted by $\mathcal{B}_i$, contains at most $N_i\leq 2^i$ sets (that we refer to as *level-i batches of terminals*), each of which contains roughly $\eta_i=\frac{|T|}{2^{\Theta(i)}}$ terminals. Additionally, every set in the level-i partition $\mathcal{B}_i$ is contained in some level-$(i-1)$ batch. We also ensure that every level-z batch $B\in\mathcal{B}_z$ contains exactly one terminal. Our algorithm consists of z phases, where, in the ith phase, we process the level-i partition $\mathcal{B}_i$ of the terminals. We partition the set $\mathcal{B}_i$ into a collection $\mathcal{B}_i^A$ of *active* level-i batches and a collection $\mathcal{B}_i^I$ of *inactive* level-i batches. Additionally, for every active batch $B\in\mathcal{B}_i^A$, we will compute a set $A_B\subseteq V(G)\setminus(T'\cup\{t^*\}\cup N^-_G(t^*))$ of vertices with $A_B\cap B\neq\emptyset$. We will ensure that, on the one hand, for every active batch $B\in\mathcal{B}_i^A$, $\mathrm{Vol}^+_G(A_B)$ is sufficiently small, while, on the other hand, if, for some level-i batch $B\in\mathcal{B}_i$, $B\cap L\neq\emptyset$ holds, then B must be an active batch, and $L\subseteq A_B$ must hold. For every level $0\leq i\leq z$, every active level-i batch $B\in\mathcal{B}_i^A$ is associated with the graph $G_B=G^{|A_B}$ — the graph derived from G via the vertex set A_B. Since $\mathrm{Vol}^+_G(A_B)$ is relatively small, so is $|E(G_B)|$.

Consider now the last level z of the hierarchical partition of the terminals. For every level-z batch $B\in\mathcal{B}_z$, $|B|=1$ holds. When processing this last level of the partition, our algorithm will compute, for each active batch $B\in\mathcal{B}_z^A$, a vertex-cut (L_B,S_B,R_B) in G, so that, if the unique terminal of B lies in L, then (L_B,S_B,R_B) is a global minimum vertex-cut. Cut (L_B,S_B,R_B) is computed by first computing a minimum vertex-cut in G_B separating the unique terminal in B from t, and then converting the resulting cut into a vertex-cut in G. The key is that, since we ensure that the sizes of the graphs in $\{G_B\}_{B\in\mathcal{B}_z}$ are sufficiently small, and each batch $B\in\mathcal{B}_z$ only contains one terminal, we can compute the cut (L_B,S_B,R_B) in time $O\left(|E(G_B)|^{1+o(1)}\right)$, and we can afford to spend this running time for all $B\in\mathcal{B}_z$. We now proceed to formally describe the preprocessing step.

5.1 The Preprocessing Step

In the preprocessing step, we construct the collections T and T' of terminals and anti-terminals, respectively, and select the vertex $t^*\in V(G)$. We also construct the hierarchical partition of

the terminals.

We start by selecting an integer $1 \leq j \leq \lceil \log m \rceil$ uniformly at random and setting $\lambda = \frac{m}{2^j}$. We say that the good event $\mathcal{E}_1$ happens if $\frac{\lambda}{2} \leq \mathrm{Vol}_G^+(L) \leq \lambda$. It is immediate to verify that $\mathbf{Pr}[\mathcal{E}_1] \geq \frac{1}{\lceil \log m \rceil} \geq \frac{1}{4 \log n}$.

Next, we select a vertex $t^* \in V(G)$ uniformly at random. We let $\mathcal{E}_2$ be the good event that $t^* \in R$. Note that, since $\hat{k} \leq \frac{n}{1000}$, if $k \leq \hat{k}$ holds, then $k \leq \frac{n}{1000}$ must hold. Since we also assumed that $|L| \leq |R|$, we get that, if $k \leq \hat{k}$, then $|R| \geq \frac{n}{4}$. Therefore, if $k \leq \hat{k}$, $\mathbf{Pr}[\mathcal{E}_2] \geq \frac{1}{4}$.

Our next step is to select a random bit $b \in \{0,1\}$ uniformly at random. We say that the good event $\mathcal{E}_3$ happens if either (i) $b = 0$ and $\mathrm{Vol}^+(L) \geq \frac{m}{4}$; or (ii) $b = 1$ and $\mathrm{Vol}^+(L) < \frac{m}{4}$. Clearly, $\mathbf{Pr}[\mathcal{E}_3] = \frac{1}{2}$.

We now consider two cases. Assume first that $b = 0$. In this case, we select a vertex $s^* \in V(G)$ at random, where, for every vertex $v \in V(G)$, the probability to select $s^* = v$ is $\frac{\deg^+(v)}{m}$. We say that the good event $\mathcal{E}'$ happens if $s^* \in L$. Note that, if Event $\mathcal{E}_3$ has happened, then $\mathrm{Vol}_G^+(L) \geq \frac{m}{4}$, and so $\mathbf{Pr}[\mathcal{E}' \mid \mathcal{E}_3] \geq \frac{1}{4}$. We then compute a minimum s^*-t^* vertex-cut (L', S', R') in G using the algorithm from Corollary 2.11 in time $O\left(m^{1+o(1)}\right)$, and return this cut as the algorithm's outcome. Note that, if $b = 0$, and if both events $\mathcal{E}_2$ and $\mathcal{E}'$ happened, then $s^* \in L$ and $t^* \in R$ must hold, and the cut (L', S', R') must be a global minimum vertex-cut. Note that, if $\mathrm{Vol}_G^+(L) \geq \frac{m}{4}$ and $k \leq \hat{k}$, then $\mathbf{Pr}[\mathcal{E}_2 \wedge \mathcal{E}_3 \wedge \mathcal{E}'] \geq \mathbf{Pr}[\mathcal{E}_2] \cdot \mathbf{Pr}[\mathcal{E}_3] \cdot \mathbf{Pr}[\mathcal{E}' \mid \mathcal{E}_3] \geq \frac{1}{32}$. Altogether, if $\mathrm{Vol}_G^+(L) \geq \frac{m}{4}$ and $k \leq \hat{k}$, then, with probability at least $\frac{1}{32}$, the algorithm described so far outputs a global minimum vertex-cut. We assume from now on that $b = 1$.

Our next step is to compute a set $T_0 \subseteq V(G)$ of vertices that we call *initial terminals*, by performing $r = \left\lceil \frac{m}{\lambda} \right\rceil$ independent trials. In every trial we select a vertex from $V(G)$, where the probability for selecting a vertex v is $\frac{\deg_G^+(v)}{m}$; the selected vertex is then added to T_0. These trials are performed independently and with repetitions. We construct a set $T' \subseteq V(G)$ of vertices, that we refer to as *anti-terminals*, using the same process. Finally, we let $T = T_0 \setminus \left(T' \cup \{t^*\} \cup N_G^-(t^*)\right)$ be the set of *terminals*. We say that a good event $\mathcal{E}$ happens if all of the following hold:

- $\frac{\lambda}{2} \leq \mathrm{Vol}_G^+(L) \leq \lambda$;
- $t^* \in R$;
- Event $\mathcal{E}_3$ happens;
- $T_0 \cap L \neq \emptyset$; and
- $T' \cap L = \emptyset$.

In the following simple observation we show that, if $\mathrm{Vol}_G^+(L) < \frac{m}{4}$ and $k \leq \hat{k}$, then Event $\mathcal{E}$ happens with a sufficiently high probability. The proof is elementary and is deferred to the full version of the paper.

Observation 5.3. *If* $\mathrm{Vol}_G^+(L) < \frac{m}{4}$ *and* $k \leq \hat{k}$, *then* $\mathbf{Pr}[\mathcal{E}] \geq \frac{1}{2^{64} \cdot \log n}$.

Throughout, we refer to the vertices in T' as *anti-terminals*, and we let $T = T_0 \setminus \left(T' \cup \{t^*\} \cup N_G^-(t^*)\right)$ be the set of vertices that we refer to as *terminals*. The vertices of T_0 are called *initial terminals* and are only used in the analysis of the preprocessing step. Note that, if Event $\mathcal{E}$ happened, then $T \cap L \neq \emptyset$ must hold.

Hierarchical Partition of the Terminals. Our algorithm computes a hierarchical partition of the terminals, as follows. The hierarchy will have $z = O(\log n)$ *levels*, and, for all $0 \leq i \leq z$, we will construct a partition $\mathcal{B}_i$ of T, that we refer to as the *level-i partition*. We refer to the sets $B \in \mathcal{B}_i$ in the level-i partition as *level-i batches of terminals*. We will denote by $\eta_i = \max_{B \in \mathcal{B}_i} \{|B|\}$ the largest cardinality of a level-i batch, and we will ensure that, for all $B \in \mathcal{B}_i$, $\frac{\eta_i}{4} \leq |B| \leq \eta_i$ holds. We will also ensure that, for $1 \leq i \leq z$, $\frac{\eta_{i-1}}{2} \leq \eta_i \leq \frac{2\eta_{i-1}}{3}$. We now provide a formal description of the construction of the hierarchical partition of the terminals.

The level-0 partition, $\mathcal{B}_0$, consists of a single set of terminals T, and we let $\eta_0 = |T|$. Consider now some level $i > 0$ and assume that the level-$(i-1)$ partition $\mathcal{B}_{i-1}$ of the terminals was computed already. Let $\eta_{i-1} = \max_{B \in \mathcal{B}_{i-1}} \{|B|\}$, and assume additionally that, for every set $B \in \mathcal{B}_{i-1}$, $|B| \geq \frac{\eta_{i-1}}{4}$ holds.

In order to compute the level-i partition, we start with $\mathcal{B}_i = \emptyset$, and then consider every level-$(i-1)$ batch $B \in \mathcal{B}_{i-1}$ one by one. For each such batch B, if $|B| = 1$ or $|B| \leq \frac{\eta_{i-1}}{2}$, then we add the set B to $\mathcal{B}_i$. Otherwise, we compute an arbitrary partition (B', B'') of B, where B' contains exactly $\left\lceil \frac{|B|}{2} \right\rceil$ and B'' exactly $\left\lfloor \frac{|B|}{2} \right\rfloor$ terminals of B, and add both B' and B'' to $\mathcal{B}_{i-1}$. This concludes the description of the algorithm for computing the level-i partition $\mathcal{B}_i$. We denote by $\eta_i = \max_{B' \in \mathcal{B}_i} \{|B'|\}$. Since, for every level-$(i-1)$ batch $B \in \mathcal{B}_{i-1}$, $\frac{\eta_{i-1}}{4} \leq |B| \leq \eta_{i-1}$ holds, it is immediate to verify that, if $\eta_{i-1} \geq 2$, then $\eta_i \leq \left\lceil \frac{\eta_{i-1}}{2} \right\rceil \leq \frac{2\eta_{i-1}}{3}$. Moreover, for every set $B' \in \mathcal{B}_{i-1}$, $|B'| \geq \frac{\eta_{i-1}}{4} \geq \frac{\eta_i}{4}$. It is also immediate to verify that $\eta_i \geq \frac{\eta_{i-1}}{2}$ must hold.

Once we reach a level i for which $\eta_i = 1$, we set $z = i$ and terminate the algorithm for constructing the hierarchical partition of the terminals. From the above discussion, $\eta_0 = |T| \leq r = \left\lceil \frac{m}{\lambda} \right\rceil$, and for all $1 \leq i \leq z$:

$$\frac{\eta_{i-1}}{2} \leq \eta_i \leq \frac{2\eta_{i-1}}{3}. \tag{5.1}$$

Therefore, for all $1 \leq i \leq z$:

$$\frac{|T|}{2^i} \leq \eta_i \leq |T| \cdot \left(\frac{2}{3}\right)^i. \tag{5.2}$$

In particular, $z \leq \frac{\log r}{\log(3/2)} \leq 2\log n$. For all $0 \leq i \leq z$, we denote $N_i = |\mathcal{B}_i|$. Since $\mathcal{B}_i$ is a partition of T, and since, for every level-i batch $B \in \mathcal{B}_i$, $\frac{\eta_i}{4} \leq |B| \leq \eta_i$ holds, we get that:

$$\frac{|T|}{\eta_i} \leq N_i \leq \frac{4|T|}{\eta_i}. \tag{5.3}$$

It is immediate to verify that the hierarchical partition of the terminals can be computed in time $O(nz) \leq O(n \log n)$, and overall, the total running time of the preprocessing step is bounded by $O(n \log n)$.

Algorithm Overview. Our algorithm consists of z phases. For all $1 \leq i \leq z$, in Phase i we compute a partition of the set $\mathcal{B}_i$ of level-i batches into the collection $\mathcal{B}_i^A$ of *active batches*, and the collection $\mathcal{B}_i^I$ of *inactive batches*. For every active batch $B \in \mathcal{B}_i^A$, we will compute a subset $A_B \subseteq V(G) \setminus \left(\{t^*\} \cup N_G^-(t^*) \cup T'\right)$ of vertices of G, with $\mathrm{Vol}_G^+(A_B) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_i}$ and $B \cap A_B \neq \emptyset$. We will ensure that, if $B \in \mathcal{B}_i$ is a level-i batch of terminals with $L \cap B \neq \emptyset$, then $B \in \mathcal{B}_i^A$ holds, and, moreover, $L \subseteq A_B$.

After the last phase is completed, for every active level-z batch $B \in \mathcal{B}_z^A$, we will compute the graph $G_B = G^{|A_B}$ that is derived from G via the vertex set A_B, and a minimum vertex-cut in G_B between the unique terminal $s \in B$ and t. We will then select the smallest-value cut among the resulting cuts, transform it into a vertex-cut in G of the same value, and output this vertex-cut.

The Main Technical Claim. At a very high level, in order to execute the ith phase, for $i > 0$, the algorithm processes each level-i batch $B \in \mathcal{B}_i$ one by one. Consider any such batch B, and let $B' \in \mathcal{B}_{i-1}$ be the unique level-$(i-1)$ batch containing B. If B' is an inactive batch, then we add B to the set $\mathcal{B}_i^I$ of inactive level-i batches. Assume now that B' is an active batch. The algorithm considers the corresponding graph $G_{B'}$ derived from G via $A_{B'}$, and then computes a vertex-cut $(\hat{L}, \hat{S}, \hat{R})$ in $G_{B'}$ with $t \in \hat{R}$, such that $T' \cap \hat{L} = \emptyset$, and $|\hat{S}| \leq 3\hat{k} \cdot \eta_i$. It then sets $A_B = \hat{L}$. Let $G_B = G^{|A_B}$ be the graph derived from G via the vertex set A_B. From the properties of graphs derived via vertex sets, $(\hat{L}, \hat{S}, V(G) \setminus (\hat{L} \cup \hat{S}))$ is a valid vertex-cut in G (see Claim 2.4). In order to ensure that $\mathrm{Vol}_G^+(A_B) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_i}$, it is enough to ensure that $\mathrm{Vol}_G^+(\hat{L}) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_i}$. Our main technical claim shows that, with a sufficiently high probability, for every vertex-cut $(\hat{L}, \hat{S}, V(G) \setminus (\hat{L} \cup \hat{S}))$ in G that may arise from the above process, $\mathrm{Vol}_G^+(\hat{L})$ must indeed be sufficiently low. In fact, the main purpose of selecting the

anti-terminals is to ensure precisely this property. We now define the notion of a *bad tripartition* and show that, with a sufficiently high probability, no such bad tripartitions of vertices of G exist.

DEFINITION 5.4 (Bad tripartition). *Consider a level $0 \leq i \leq z$ and a tripartition (L', S', R') of $V(G)$. We say that (L', S', R') is a* bad tripartition for level i *if all of the following hold:*

- *B1.* $|S'| \leq 3\hat{k} \cdot \eta_i$;
- *B2.* $L' \cap T' = \emptyset$;
- *B3.* $\mathrm{Vol}^+_G(L') \geq \frac{1000\hat{k} \cdot m \cdot \log n}{N_i}$; *and*
- *B4. there is a collection $\hat{B} \subseteq T \setminus S'$ of at most η_i terminals, such that L' is precisely the collection of all vertices of G that are reachable from the vertices of $\hat{B}$ in $G \setminus S'$. In other words, for every vertex $v \in L'$, there is a path from some vertex $u \in \hat{B}$ to v in G that avoids the vertices of S', and all vertices of G with this property lie in L'.*

We let $\hat{\mathcal{E}}$ be the bad event that, for some level $0 \leq i \leq z$, a bad tripartition of $V(G)$ for level i exists. The following technical claim, whose proof is deferred to the full version of the paper, is key to our algorithm.

CLAIM 5.5. $\mathbf{Pr}\left[\hat{\mathcal{E}}\right] \leq \frac{1}{n^2}$.

We let $\mathcal{E}^*$ be the good event that the Event $\mathcal{E}$ has happened and the Event $\hat{\mathcal{E}}$ did not happen. By combining Observation 5.3 with Claim 5.5 and the Union Bound, we get that, if $\mathrm{Vol}^+_G(L) < \frac{m}{4}$ and $k \leq \hat{k}$, then:

$$\mathbf{Pr}\left[\neg\mathcal{E}^*\right] \leq \mathbf{Pr}\left[\neg\mathcal{E}\right] + \mathbf{Pr}\left[\hat{\mathcal{E}}\right] \leq 1 - \frac{1}{2^{64} \cdot \log n} + \frac{1}{n^2} \leq 1 - \frac{1}{2^{65} \cdot \log n}.$$

Therefore, if $\mathrm{Vol}^+_G(L) < \frac{m}{4}$ and $k \leq \hat{k}$, then:

$$\mathbf{Pr}\left[\mathcal{E}^*\right] \geq \frac{1}{2^{65} \cdot \log n}. \tag{5.4}$$

For convenience, in the remainder of the proof, we denote by $T'' = T' \cup N^-_G(t^*) \cup \{t^*\}$. Note that $T'' \cap T = \emptyset$ must hold, and, if Event $\mathcal{E}^*$ happened, then $T'' \cap L = \emptyset$.

Processing a Single Batch of Terminals The key subroutine of our algoritm is a procedure for processing a single batch of terminals, that is summarized in the following claim, whose proof is deferred to the full version of the paper.

CLAIM 5.6. *There is a deterministic algorithm, whose input consists of an integer $1 \leq i \leq z$, a level-$(i-1)$ batch $B \in \mathcal{B}_{i-1}$ of terminals, a set $A_B \subseteq V(G) \setminus T''$ of vertices of G with $B \cap A_B \neq \emptyset$ and $\mathrm{Vol}^+_G(A_B) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_{i-1}}$, together with a level-$i$ batch $B' \subseteq B$ of terminals. The algorithm either returns "FAIL", or computes a set $A_{B'} \subseteq V(G) \setminus T''$ of vertices of G with $B' \cap A_{B'} \neq \emptyset$ and $\mathrm{Vol}^+_G(A_{B'}) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_i}$. The algorithm guarantees that, if all of the following hold:*

- *Event $\mathcal{E}^*$ has happened;*
- *$k \leq \hat{k}$;*
- *$L \subseteq A_B$; and*
- *$B' \cap L \neq \emptyset$,*

then it may not return "FAIL", and moreover, $L \subseteq A_{B'}$ must hold. The running time of the algorithm is $O\left(\frac{\hat{k} \cdot m^{1+o(1)}}{N_{i-1}}\right)$.

We now complete the proof of Theorem 5.2.

The Algorithm Description At the beginning of the algorithm, we consider the unique level-0 batch $B \in \mathcal{B}_0$ of terminals (recall that $B = T$). We set $\mathcal{B}_0^I = \emptyset$ and $\mathcal{B}_0^A = \mathcal{B}_0 = \{B\}$. We also define $A_B = V(G) \setminus T''$. Since $N_0 = 1$, it is immediate to verify that $\mathrm{Vol}^+(A_B) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_0}$. Assume now that Event $\mathcal{E}^*$ has happened, so, in particular, $L \cap B = L \cap T \neq \emptyset$ and $t^* \in R$ hold. Then $N_G^-(t^*) \subseteq S \cup R$ must hold, and so $L \subseteq A_B$. Next, we perform at most z phases.

Execution of a Phase. We now fix an index $1 \leq i \leq z$ and describe the execution of the ith phase. We assume that, at the beginning of Phase i, we are given a partition $(\mathcal{B}_{i-1}^A, \mathcal{B}_{i-1}^I)$ of the set $\mathcal{B}_{i-1}$ of level-$(i-1)$ batches, where the batches in $\mathcal{B}_{i-1}^A$ are referred to as *active*, and the batches in $\mathcal{B}_{i-1}^I$ as *inactive*. We also assume that, for every active level-$(i-1)$ batch $B \in \mathcal{B}_{i-1}^A$, we are given a set $A_B \subseteq V(G) \setminus T''$ of vertices of G, such that $B \cap A_B \neq \emptyset$ and $\mathrm{Vol}_G^+(A_B) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_{i-1}}$ hold. Lastly, we assume that, if Event $\mathcal{E}^*$ has happened and $k \leq \hat{k}$ then for every level-$(i-1)$ batch $B \in \mathcal{B}_{i-1}$, if $L \cap B \neq \emptyset$, then $B \in \mathcal{B}_{i-1}^A$ holds, and, moreover, $L \subseteq A_B$.

We consider every level-i batch one by one. Let $B' \in \mathcal{B}_i$ be any such batch, and let $B \in \mathcal{B}_{i-1}$ be the unique level-$(i-1)$ batch with $B' \subseteq B$. If $B \in \mathcal{B}_{i-1}^I$, then we add B' to the set $\mathcal{B}_i^I$ of inactive level-i batches. Otherwise, we apply the algorithm from Claim 5.6 to the batches B and B' and the set A_B of vertices of G. If the algorithm returns "FAIL", then we add B' to the set $\mathcal{B}_i^I$ of inactive level-i batches. Otherwise, the algorithm must have computed a set $A_{B'} \subseteq V(G) \setminus T''$ of vertices of G with $B' \cap A_{B'} \neq \emptyset$ and $\mathrm{Vol}_G^+(A_{B'}) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_i}$. We then add B' to the set $\mathcal{B}_i^A$ of active level-i batches, and we record its associated vertex set $A_{B'}$.

This completes the description of the algorithm for Phase i. Note that, at the end of the algorithm, we obtain a partition $(\mathcal{B}_i^A, \mathcal{B}_i^I)$ of the set $\mathcal{B}_i$ of level-i batches into the set $\mathcal{B}_i^A$ of active batches and the set $\mathcal{B}_i^I$ of inactive batches. For every active level-i batch $B' \in \mathcal{B}_i^A$, we have computed a set $A_{B'} \subseteq V(G) \setminus T''$ of vertices of G with $B' \cap A_{B'} \neq \emptyset$ and $\mathrm{Vol}_G^+(A_{B'}) \leq \frac{1000\hat{k} \cdot m \cdot \log n}{N_i}$. Lastly, assume that Event $\mathcal{E}^*$ has happened and that $k \leq \hat{k}$, and consider some level-i batch $B' \in \mathcal{B}_i$ with $B' \cap L \neq \emptyset$. Let $B \in \mathcal{B}_i$ be the unique level-$(i-1)$ batch containing B. Then, from our assumption, $B \in \mathcal{B}_{i-1}^A$ and $L \subseteq A_B$ hold. Therefore, our algorithm must have applied the algorithm from Claim 5.6 to the batches B and B', and, moreover, the algorithm from Claim 5.6 did not return "FAIL". Therefore, B' was added to the set $\mathcal{B}_i^A$ of active level-i batches. Lastly, the algorithm from Claim 5.6 guarantees that in this case, if $A_{B'}$ is the vertex set that it returns, then $L \subseteq A_{B'}$.

Recall that the running time of the algorithm from Claim 5.6 is $O\left(\frac{\hat{k} \cdot m^{1+o(1)}}{N_{i-1}}\right)$, and we apply it to at most $|\mathcal{B}_i| \leq N_i$ batches. Moreover, from Inequalities 5.3 and 5.1, $N_i \leq O(N_{i-1})$. Therefore, the running time of a single phase is:

$$O\left(\frac{\hat{k} \cdot m^{1+o(1)}}{N_{i-1}} \cdot N_i\right) \leq O\left(\hat{k} \cdot m^{1+o(1)}\right).$$

Completing the Algorithm. Recall that every level-z batch $B \in \mathcal{B}_z$ of terminals contains exactly one terminal, that we denote by s_B. Recall also that, if Event $\mathcal{E}^*$ has happened, then $T \cap L \neq \emptyset$. If Event $\mathcal{E}^*$ has happened, then we let s^* be any terminal in $T \cap L$, and otherwise we let s^* be any terminal in T. We denote by B^* the unique level-z batch with $s_{B^*} = s^*$. Note that, if Event $\mathcal{E}^*$ has happened and $k \leq \hat{k}$, then our algorithm guarantees that $B^* \in \mathcal{B}_z^A$ holds, and, moreover, $L \subseteq A_{B^*}$.

If $\mathcal{B}_z^A = \emptyset$, then we terminate the algorithm and return the arbitrary vertex-cut $(\tilde{L}, \tilde{S}, \tilde{R})$ that we computed at the beginning of the algorithm; note that, from our discussion, this may only happen if Event $\mathcal{E}^*$ did not happen, or if $\hat{k} > k$.

Assume now that $\mathcal{B}_z^A \neq \emptyset$. We process every active level-z batch $B \in \mathcal{B}_z^A$ one by one. When a batch $B \in \mathcal{B}_z^A$ is processed, we compute the corresponding graph $G_B = G^{|A_B}$ that is derived from G via A_B. Recall that, from Claim 2.3, graph G_B can be computed in time $O\left(\mathrm{Vol}_G^+(A_B)\right) \leq O\left(\frac{\hat{k} \cdot m \cdot \log n}{N_z}\right)$, and, moreover, $|E(G_B)| \leq O\left(\mathrm{Vol}_G^+(A_B)\right) \leq O\left(\frac{\hat{k} \cdot m \cdot \log n}{N_z}\right)$. Recall also that $A_B \cap B \neq \emptyset$ must hold, so $s_B \in A_B$. We then compute a minimum s_B-t vertex-cut (L_B, S_B, R_B) in G_B in time $O\left(|E(G_B)|^{1+o(1)}\right) \leq O\left(\frac{\hat{k} \cdot m^{1+o(1)}}{N_z}\right)$, using the algorithm from Corollary 2.11. Finally, we select a batch $B \in \mathcal{B}_z$ for which $|S_B|$ is minimized, and return the tripartition

(L', S', R') of $V(G)$ that is induced by the vertex set $L' = L_B$. Recall that, from Claim 5.6, $A_B \subseteq V(G) \setminus T''$ must hold, so $t^* \notin A_B \cup N^+_G(A_B)$, and, in particular, $A \cup N^+_G(A) \neq V(G)$. From Claim 2.4, (L', S', R') is a valid vertex-cut in G. We return this cut as the algorithm's output.

From the above discussion, the time required to process a single level-z batch $B \in \mathcal{B}^A_z$ is bounded by $O\left(\frac{\hat{k} \cdot m^{1+o(1)}}{N_z}\right)$, and, since the number of level-z batches is bounded by N_z, the running time of this last step is bounded by $O\left(\hat{k} \cdot m^{1+o(1)}\right)$. Since the algorithm has $z = O(\log n)$ phases, and the running time of each phase is bounded by $O\left(\hat{k} \cdot m^{1+o(1)}\right)$, and since the running time of the preprocessing step is bounded by $O(m)$, the total running time of the algorithm is $O\left(\hat{k} \cdot m^{1+o(1)}\right)$.

As observed already, if Event $\mathcal{E}^*$ happened and $\hat{k} \leq k$, then $B^* \in \mathcal{B}^A_z$ and $L \subseteq A_{B^*}$ hold. From Claim 2.3, $(L, S, V(G_{B^*}) \setminus (L \cup S)))$ is a valid vertex-cut in G_{B^*}, and it is easy to see that it must be an s^*-t vertex-cut in G_{B^*} of value k. In this case, we are guaranteed that the vertex-cut (L', S', R') that our algorithm returns has value $|S'| = k$.

Recall that, if $\mathrm{Vol}^+_G(L) \geq \frac{m}{4}$, then the vertex-cut that our algorithm returns is guaranteed to be optimal with probability at least $\frac{1}{32}$. If $\mathrm{Vol}^+_G(L) < \frac{m}{4}$, and if $\hat{k} \leq k$ and Event $\mathcal{E}^*$ happens, then the vertex-cut that our algorithm returns is guaranteed to be optimal. Since, from Equation (5.4), if $k \leq \hat{k}$ and $\mathrm{Vol}^+_G(L) < \frac{m}{4}$ then $\mathbf{Pr}\left[\mathcal{E}^*\right] \geq \frac{1}{2^{65} \cdot \log n}$, we get that, overall, if $\hat{k} \leq k$, the probability that the vertex-cut that our algorithm returns is optimal is $\Omega\left(\frac{1}{\log n}\right)$. By repeating the algorithm $O(\log^2 n)$ times and returning the lowest-value cut among the resulting vertex-cuts, we guarantee that the probability that the vertex-cut that our algorithm returns is optimal is at least $\left(1 - \frac{1}{n}\right)$. The running time of the algorithm remains $O\left(\hat{k} \cdot m^{1+o(1)}\right)$.

6 Algorithm for Weighted Directed Non-Dense Graphs

In this section we provide a randomized algorithm for the weighted directed global minimum vertex-cut problem with running time $O\left(mn^{11/12+o(1)} \cdot d^{1/12} \cdot \log^2 W\right)$, where d is the average vertex degree in the input graph. The algorithm is most suitable for graphs that are not too dense, and is summarized in the following theorem.

THEOREM 6.1. *There is a randomized algorithm, whose input consists of a simple directed n-vertex and m-edge graph G with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V(G)$, such that G contains some vertex-cut. The algorithm computes a vertex-cut (L', S', R') in G, so that, with probability at least $\frac{1}{2}$, (L', S', R') is a global minimum vertex-cut. The running time of the algorithm is $O\left(mn^{11/12+o(1)} \cdot d^{1/12} \cdot \log^2(W)\right)$, where d is the average vertex degree in G.*

In the remainder of this section we prove Theorem 6.1. We will use a parameter $0 < \epsilon < 1$, whose value will be set later. If there is a global minimum vertex-cut (L, S, R) in the input graph G for which either $|L| \geq n^\epsilon$ or $\mathrm{Vol}_G(L) \geq n^\epsilon \cdot d$ hold, where d is the average vertex degree in G, we will employ Algorithm AlgSpecial from Theorem 3.1 to compute a global minimum vertex-cut in G. Otherwise, we are guaranteed that every global minimum vertex-cut (L, S, R) in G has properties P1–P3. It is now sufficient to design an algorithm for this latter case.

In the remainder of this section, we assume that we are given a directed n-vertex and m-edge graph $G = (V, E)$. Using the transformation from Subsection 2.3, we can assume that, for every vertex $v \in V(G)$, its weight is an integer $w(v) \geq 1$. We also assume that we are given a parameter $0 < \epsilon < 1$, whose value we will set later. For the sake of the analysis, we fix a global minimum vertex-cut (L, S, R) in G, that we refer to as the *distinguished min-cut*, as follows. If there exists a global minimum vertex-cut in G for which Properties P1-P3 hold, then we let (L, S, R) be any such cut, and we say that the distinguished min-cut is *good*; otherwise, we let (L, S, R) be an arbitrary global minimum vertex-cut and we say that the distinguished min-cut is *bad*. As before, we use the convention that $w(L) \leq w(R)$ holds. Throughout, we denote by d the average vertex degree in G. We also denote the parameters $W'_{\max}(G)$ and $W_{\max}(G)$ by $W'_{\max}$ and $W_{\max}$ respectively (see Subsection 2.4 for a definition).

Critical Threshold. One of the central notions that our algorithm uses is that of a *critical threshold*, that we define next.

DEFINITION 6.2 (Critical Threshold). *Let G be a directed graph with integral weights $w(v) \geq 1$ on its vertices $v \in V(G)$, let $0 < \epsilon < 1$ be a parameter, and let (L', S', R') be a global minimum vertex-cut in G. An integer τ^* is the* critical threshold *for (L', S', R'), if it is the smallest integral power of 2 with $\tau^* \geq 1$, such that every vertex $u \in S'$ with $w(u) > \tau^*$ is a high-degree vertex (see Definition 4.1). Notice that it is possible that S' contains no vertex u with $w(u) > \tau^*$.*

Observe that $1 \leq \tau^* \leq W'_{\max}$ must hold. Moreover, if $\tau^* = 1$ then it is possible that S only contains high-degree vertices. The critical thershold τ^* is not known to the algorithm. However, we will supply the algorithm with a *guess* τ on the value of the critical threshold. We will then require that, if $\tau = \tau^*$, then the algorithm returns a global minimum vertex-cut in G with a sufficiently high probability.

Selecting parameters λ, τ and a set Γ of vertex pairs. As our first step, we select values $1 \leq \lambda \leq n$ and $1 \leq \tau \leq W'_{\max}$, that are both integral powers of 2, together with a set Γ of at most $\left\lceil \frac{100n \log n}{\lambda} \right\rceil$ pairs of vertices of G. Our goal is to ensure that, with a sufficiently high probability, $\tau = \tau^*$, $\frac{\lambda}{2} \leq |L| \leq \lambda$, and there is a pair $(x, y) \in \Gamma$ with $x \in L$ and $y \in R$ (recall that (L, S, R) is the distingiushed min-cut). The algorithm is summarized in the next claim, whose proof is deferred to the full version of the paper.

CLAIM 6.3. *There is a randomized algorithm, that we call* AlgSelectPairs*, whose input consists of a directed n-vertex and m-edge graph G with integral weights $w(v) \geq 1$ on its vertices $v \in V(G)$ and a parameter $0 < \epsilon < 1$. The algorithm returns values $1 \leq \tau \leq W'_{\max}$ and $1 \leq \lambda \leq 2n^{\epsilon}$ that are integral powers of 2, together with a set Γ of at most $\left\lceil \frac{100n \log n}{\lambda} \right\rceil$ pairs of vertices of G, such that, for every pair $(x, y) \in \Gamma$, $x \neq y$ and $(x, y) \notin E(G)$. We say that the algorithm is* successful *with respect to a fixed global minimum vertex-cut (L, S, R), if $\frac{\lambda}{2} \leq |L| \leq \lambda$, $\tau = \tau^*$, where τ^* is the critical threshold for (L, S, R), and there is a pair $(x, y) \in \Gamma$ with $x \in L$ and $y \in R$. If (L, S, R) is a global minimum vertex-cut for which Properties P1–P3 hold, then the algorithm is successful with respect to this cut with probability at least $\frac{1}{32 \log n \cdot \log W_{\max}}$. The running time of the algorithm is $\tilde{O}(m \cdot n^{\epsilon})$.*

In the remainder of the proof, we denote by $\mathcal{E}_1$ the good event that Algorithm AlgSelectPairs was successful with respect to the distinguished min-cut (L, S, R). From Claim 6.3, if the distinguished cut was good, then $\mathbf{Pr}[\mathcal{E}_1] \geq \frac{1}{32 \log n \cdot \log(W_{\max})}$.

For the sake of analysis, we designate a single pair $(x^*, y^*) \in \Gamma$ as the *distinguished pair* as follows: if Algorithm AlgSelectPairs was successful, then $(x^*, y^*) \in \Gamma$ is an arbitrary pair with $x^* \in L$ and $y^* \in R$. Otherwise, (x^*, y^*) is an arbitrary pair in Γ.

Two Special Cases. Throughout, we denote by $\tau' = \frac{\tau}{64\lambda^2}$. Recall that we denoted by d the average vertex degree in G, and that a vertex $v \in V(G)$ is *high-degree* iff $\deg_G(v) \geq d \cdot n^{\epsilon}$. We denote by V^{hd} and V^{ld} the sets of all high-degree and all low-degree vertices of G respectively. We also denote by $V^{\geq \tau'} = \{v \in V(G) \mid w(v) \geq \tau'\}$. Consider now the distinguished min-cut (L, S, R). Throughout, we denote by $\hat{S} \subseteq S$ the set of vertices containing all low-degree vertices of S whose weight is at least τ'. In other words:

$$\hat{S} = S \cap V^{\mathrm{ld}} \cap V^{\geq \tau'}.$$

Next we will consider two cases: Case 1 happens if $|\hat{S}| \leq 2^{12}\lambda^3$ and Case 2 happens otherwise. We summarize our algorithms for each of the two cases in the following two lemmas. The statements of the two lemmas are essentially identical, except that the first one requires that $|\hat{S}| \leq 2^{12}\lambda^3$, while the second requires that $|\hat{S}| > 2^{12}\lambda^3$, and they differ in the running times of their respective algorithms.

LEMMA 6.4 (Algorithm for Case 1). *There is a randomized algorithm, that is given as input a directed n-vertex and m-edge graph $G = (V, E)$ with integral weights $w(v) \geq 1$ on its vertices $v \in V$, a set $\Gamma \subseteq V$ of at most $\left\lceil \frac{100n \log n}{\lambda} \right\rceil$ pairs of vertices, such that, for every pair $(x, y) \in \Gamma$, $x \neq y$ and $(x, y) \notin E(G)$ hold, a parameter $0 < \epsilon < 1$, and two additional parameters $1 \leq \lambda \leq 2n^{\epsilon}$ and $1 \leq \tau \leq W_{\max}$, both integral powers of 2. The*

algorithm returns a vertex-cut (L', S', R') in G. The algorithm guarantees that, if there exists a global minimum vertex-cut (L, S, R) in G for which all of the following hold:

- *Properties P1-P3 hold for (L, S, R);*
- *$\frac{\lambda}{2} \leq |L| \leq \lambda$;*
- *$\tau = \tau^*$, where τ^* is the critical threshold for (L, S, R);*
- *$|\hat{S}| \leq 2^{12}\lambda^3$, where $\hat{S} = S \cap V^{\mathrm{ld}} \cap V^{\geq \tau'}$ and $\tau' = \frac{\tau}{64\lambda^2}$; and*
- *there is a pair $(x^*, y^*) \in \Gamma$ with $x^* \in L$ and $y^* \in R$,*

then, with probability at least $\frac{1}{4}$, the vertex cut (L', S', R') is a global minimum vertex-cut. The running time of the algorithm is $O\left((m \cdot n^{1-\epsilon+o(1)} + md \cdot n^{11\epsilon+o(1)}) \cdot \log(W_{\max})\right)$, where $d = \frac{2m}{n}$ is the average vertex degree in G.

LEMMA 6.5 (Algorithm for Case 2). *There is a randomized algorithm, that is given as input a directed n-vertex and m-edge graph $G = (V, E)$ with integral weights $w(v) \geq 1$ on its vertices $v \in V$, a set $\Gamma \subseteq V$ of at most $\left\lceil \frac{100n \log n}{\lambda} \right\rceil$ pairs of vertices, such that, for every pair $(x, y) \in \Gamma$, $x \neq y$ and $(x, y) \notin E(G)$ hold, a parameter $0 < \epsilon < 1$, and two additional parameters $1 \leq \lambda \leq 2n^{\epsilon}$ and $1 \leq \tau \leq W_{\max}$, both integral powers of 2. The algorithm returns a vertex-cut (L', S', R') in G. The algorithm guarantees that, if there exists a global minimum vertex-cut (L, S, R) in G for which all of the following hold:*

- *Properties P1-P3 hold for (L, S, R);*
- *$\frac{\lambda}{2} \leq |L| \leq \lambda$;*
- *$\tau = \tau^*$, where τ^* is the critical threshold for (L, S, R);*
- *$|\hat{S}| > 2^{12}\lambda^3$, where $\hat{S} = S \cap V^{\mathrm{ld}} \cap V^{\geq \tau'}$ and $\tau' = \frac{\tau}{64\lambda^2}$; and*
- *there is a pair $(x^*, y^*) \in \Gamma$ with $x^* \in L$ and $y^* \in R$,*

then, with probability at least $\frac{1}{4}$, the vertex cut (L', S', R') is a global minimum vertex-cut. The running time of the algorithm is $O\left((m \cdot n^{1-\epsilon+o(1)} + md \cdot n^{3\epsilon+o(1)}) \cdot \log(W_{\max})\right)$, where $d = \frac{2m}{n}$ is the average vertex degree in G.

The proofs of Lemmas 6.4 and 6.5 are deferred to the full version of the paper. We now complete the proof of Theorem 6.1 using them.

We set $\epsilon = \frac{1}{12} - \frac{\log d}{12 \log n}$, where d is the average vertex degree in the input graph G. We apply the algorithm AlgSpecial from Theorem 3.1 to compute, in time $O(mn^{1-\epsilon+o(1)} \log W)$, a vertex-cut (L_1, S_1, R_1) in G. Then we use Algorithm AlgSelectPairs from Claim 6.3 to compute, in time $\tilde{O}(m \cdot n^{\epsilon})$, values $1 \leq \tau \leq W'_{\max}$ and $1 \leq \lambda \leq 2n^{\epsilon}$ that are integral powers of 2, together with a set Γ of at most $\left\lceil \frac{100n \log n}{\lambda} \right\rceil$ pairs of vertices of G. We let $\mathcal{E}_1$ be the good event that the algorithm from Claim 6.3 is successful with respect to the distinguished min-cut. Lastly, we apply the algorithms from Lemmas 6.4 and 6.5 to the input graph G, the set Γ of vertex pairs, and the values λ and τ computed by Algorithm AlgSelectPairs; we denote the cuts returned by these algorithms by (L_2, S_2, R_2) and (L_3, S_3, R_3), respectively. We output the smallest-value cut from among (L_1, S_1, R_1), (L_2, S_2, R_2), and (L_3, S_3, R_3). Recall that, if either $|L| \geq n^{\epsilon}$ or $\mathrm{Vol}_G(L) \geq n^{\epsilon} \cdot d$ hold, then the cut (L_1, S_1, R_1) that Algorithm AlgSpecial returns is guaranteed to be optimal with probability at least $(1 - 1/n^2)$, and in this case, the cut that our algorithm returns is optimal as well. Otherwise, Properties P1-P3 hold for the distinguished min-cut. Fix any such cut (L, S, R). If Event $\mathcal{E}_1$ happens and $|\hat{S}| \leq 2^{12}\lambda^3$, then, from Lemma 6.4, with probability at least $\frac{1}{4}$, cut (L_2, S_2, R_2) is optimal, and in this case, the cut the our algorithm returns is optimal as well. If Event $\mathcal{E}_1$ happens and $|\hat{S}| > 2^{12}\lambda^3$, then, from Lemma 6.5, with probability at least $\frac{1}{4}$, cut (L_3, S_3, R_3) is optimal, and in this case, the cut the our algorithm returns is optimal as well. Since, from Claim 6.3, $\mathbf{Pr}[\mathcal{E}_1] \geq \frac{1}{32 \log n \cdot \log(W_{\max})}$, we get

that the probability that the cut returned by our algorithm is optimal is at least $\Omega\left(\frac{1}{\log n \log(W_{\max})}\right)$. By repeating the algorithm $O(\log n \log(W_{\max}))$ times and returning the smallest-value cut among the resulting vertex-cuts, we ensure that the cut that the algorithm returns is optimal with probability at least $\frac{1}{2}$.

Note that the running time of the algorithm is bounded by:

$$O\left((m \cdot n^{1-\epsilon+o(1)} + md \cdot n^{11\epsilon+o(1)}) \cdot \log^2(W_{\max})\right) \leq O\left(mn^{11/12+o(1)} \cdot d^{1/12} \cdot \log^2 W\right),$$

since $\epsilon = \frac{1}{12} - \frac{\log d}{12 \log n}$ and $W_{\max} \leq O(W \cdot n)$.

7 Algorithm for Weighted Directed Dense Graphs In this section we provide an algorithm for weighted directed global minimum vertex-cut that is summarized in the following theorem.

THEOREM 7.1. *There is a randomized algorithm, whose input consists of a simple directed n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W$ on its vertices $v \in V(G)$, such that G contains some vertex-cut. The algorithm computes a vertex-cut (L', S', R') in G, and it guarantees that, if there exists a global minimum vertex-cut (L, S, R) in G with $|L| \leq n^{1/2}$, then, with probability at least $\frac{1}{48 \log n}$, (L', S', R') is a global minimum vertex-cut. The running time of the algorithm is $O\left(n^{2+(5-\omega)/(11-3\omega)+o(1)} \cdot (\log W)^{O(1)}\right) \leq O\left(n^{2.677} \cdot (\log W)^{O(1)}\right)$, where ω is the matrix multiplication exponent.*

We provide the proof of Theorem 7.1 below, after we complete the proof of Theorem 1.1 using it.

7.1 Completing the Proof of Theorem 1.1 Recall that we are given as input a simple directed n-vertex and m-edge graph G with integral weights $0 \leq w(v) \leq W$ on its vertices $v \in V(G)$, such that G contains some vertex-cut. Let $d = \frac{2m}{n}$ denote the average vertex degree in G. Our goal is to compute a vertex-cut (L', S', R') in G, so that, with probability at least $1/2$, (L', S', R') is a global minimum vertex-cut in G. We design a randomized algorithm with these properties, whose running time is bounded by:

$$O\left(\min\left\{mn^{11/12+o(1)}d^{1/12}, n^{2+(5-\omega)/(11-3\omega)+o(1)}\right\}(\log W)^{O(1)}\right),$$

where ω is the matrix multiplication exponent. Using the current bound $\omega \leq 2.371552$ [37], we get that this running time is bounded by $O\left(\min\left\{mn^{11/12+o(1)} \cdot d^{1/12}, n^{2.677}\right\} \cdot (\log W)^{O(1)}\right)$, as required.

Our algorithm distinguishes between two cases. The first case happens if $mn^{11/12} \cdot d^{1/12} \leq n^{2+(5-\omega)/(11-3\omega)}$. In this case we apply the algorithm from Theorem 6.1 to the graph G, and return the vertex-cut (L', S', R') that the algorithm outputs. From Theorem 6.1, with probability at least $\frac{1}{2}$, (L', S', R') is a global minimum vertex-cut. Clearly, the running time of the algorithm in this case is bounded by:

$$O\left(mn^{11/12+o(1)} \cdot d^{1/12} \cdot \log^2(W)\right) \leq O\left(\min\left\{mn^{11/12+o(1)}d^{1/12}, n^{2+(5-\omega)/(11-3\omega)+o(1)}\right\}(\log W)^{O(1)}\right),$$

as required. From now on we assume that the second case happens, that is, $n^{2+(5-\omega)/(11-3\omega)} < mn^{11/12} \cdot d^{1/12}$. Notice that in this case, it is sufficient to design an algorithm whose running time is bounded by $O\left(n^{2+(5-\omega)/(11-3\omega)+o(1)} \cdot (\log W)^{O(1)}\right)$.

Note that, using the transformation described in Subsection 2.3, it is sufficient to design an algorithm for the special case where all vertex weights are strictly positive (since this transformation inflates the vertex weights by $O(n^2)$ factor, this will only lead to an $O(\text{poly} \log n)$ multiplicative overhead in the running time). Therefore, we assume from now on that all vertex weights in the graph G are strictly positive.

Consider the following algorithm: we apply the algorithm from Theorem 7.1 to the graph G, and denote by (L', S', R') the vertex-cut that it returns. We then apply the algorithm from Theorem 3.1 to the graph G, with the parameter $\epsilon = 1/2$, and denote by (L'', S'', R'') the vertex-cut that it returns. We then return the smaller-value vertex-cut from among the cuts (L', S', R') and (L'', S'', R''). Recall that the running time of the algorithm from

Theorem 7.1 is $O\left(n^{2+(5-\omega)/(11-3\omega)+o(1)} \cdot (\log W)^{O(1)}\right)$, while the running time of the algorithm from Theorem 3.1 is $O\left(mn^{1-\epsilon+o(1)} \log W\right) \leq O\left(mn^{1/2+o(1)} \log W\right) \leq O\left(n^{5/2+o(1)} \log W\right)$; notice that this expression is bounded by $O\left(n^{2+(5-\omega)/(11-3\omega)+o(1)} \cdot (\log W)^{O(1)}\right)$ for any possible value of $2 \leq \omega \leq 3$.

For the sake of analysis, we fix an arbitrary global minimum vertex-cut (L, S, R), that we refer to as the *distinguished cut*. From Theorem 7.1, if $|L| \leq n^{1/2}$, then the cut (L', S', R') is guaranteed to be a global minimum vertex-cut with probability at least $\frac{1}{48 \log n}$. Otherwise, from Theorem 3.1, the cut (L'', S'', R'') is guaranteed to be a global minimum vertex-cut with probability at least $\left(1 - 1/n^2\right)$. Therefore, in either case, with probability at least $\frac{1}{48 \log n}$, the cut that the algorithm returns is a global minimum vertex-cut. By repeating this algorithm $O(\log n)$ times and returning the smallest-value vertex-cut among the resulting cuts, we can guarantee that, with probability at least $\frac{1}{2}$, the cut that the algorithm returns is a global minimum vertex-cut. The total running time of the algorithm for Case 2 remains bounded by:

$$O\left(n^{2+(5-\omega)/(11-3\omega)+o(1)} \cdot (\log W)^{O(1)}\right),$$

and the total running time of the algorithm is bounded by:

$$O\left(\min\left\{mn^{11/12+o(1)} \cdot d^{1/12}, n^{2.677}\right\} (\log W)^{O(1)}\right) \leq O\left(mn^{0.976} (\log W)^{O(1)}\right).$$

In order to complete the proof of Theorem 1.1, it is now enough to prove Theorem 7.1, which we in the remainder of this section.

7.2 Proof of Theorem 7.1 Our proof follows the high-level approach from [8], but we extend their algorithm to directed graphs, while also significantly simplifying it and achieving a faster running time.

Recall that the algorithm receives as input a simple directed n-vertex and m-edge graph G with integral weights $1 \leq w(v) \leq W$ on its vertices $v \in V(G)$, such that G contains some vertex-cut. We can assume that n is greater than a large enough constant, since otherwise we can use, for example, the algorithm from Theorem 6.1, whose running time is bounded by $O(n^3 \log^2 W) \leq O(\log^2 W)$. We can also assume without loss of generality that $W = \max_{v \in V(G)} \{w(v)\}$, since otherwise we can reduce the value of W to be equal to $\max_{v \in V(G)} \{w(v)\}$, and it is sufficient to prove Theorem 7.1 with this new value of the parameter W. For the sake of the analysis, we fix a global minimum vertex-cut (L, S, R), that we refer to as the *distinguished cut*, as follows. If there exists a global minimum vertex-cut (L^*, S^*, R^*) with $|L^*| \leq n^{1/2}$, then we let (L, S, R) be any such cut, and we say that the distinguished cut is *good*. Otherwise, we let (L, S, R) be any global minimum vertex-cut, and we say that the distinguished cut is *bad*.

Recall that our goal is to compute a vertex-cut (L', S', R') in G, such that, if the distinguished cut is good, then, with probability at least $\frac{1}{48 \log n}$, (L', S', R') is a global minimum vertex-cut.

At a very high level, our algorithm follows a rather standard approach. We start by employing a simple randomized algorithm to select a value $1 \leq \lambda \leq n$ and a collection Γ of pairs of vertices of G. Intuitively, we would like to ensure that λ is close to $|L|$ (where (L, S, R) is the distinguished cut), and that there is some pair $(x, y) \in \Gamma$ with $x \in L$ and $y \in R$. We would also like to ensure that $|\Gamma|$ is not too large. We will then compute, for every pair $(x, y) \in \Gamma$, an x-y vertex-cut $(L_{x,y}, S_{x,y}, R_{x,y})$ in G, so that, if $|L| \leq \lambda$, $x \in L$ and $y \in R$ hold, then, with a sufficiently high probability, the corresponding vertex-cut $(L_{x,y}, S_{x,y}, R_{x,y})$ is a global minimum vertex-cut. We note that a similar high-level approach has been used in numerous prior works. The main difficulty with this approach is in designing an algorithm that computes, for every pair $(x, y) \in \Gamma$ of vertices, the vertex-cut $(L_{x,y}, S_{x,y}, R_{x,y})$ with the desired properties efficiently; we note that we cannot afford to spend $\Theta(m)$ time per pair on such an algorithm.

We start with the following simple claim that provides an algorithm for computing the parameter λ and the set Γ of pairs of vertices of G. The claim is similar to Claim 6.3, and its proof is deferred to the full version of the paper.

CLAIM 7.2. *There is a randomized algorithm, that we call* AlgSelectPairs$'$*, whose input consists of an n-vertex directed graph G with integral weights $w(v) \geq 1$ on its vertices $v \in V(G)$. The algorithm returns a value $1 \leq \lambda \leq n$*

that is an integral power of 2*, together with a set* Γ *of at most* $\left\lceil \frac{100n \log n}{\lambda} \right\rceil$ *pairs of vertices of* G*. We say that the algorithm is* successful *with respect to a fixed global minimum vertex-cut* (L, S, R)*, if* $\frac{\lambda}{2} \leq |L| \leq \lambda$*, and there is a pair* $(x, y) \in \Gamma$ *with* $x \in L$ *and* $y \in R$*. Let* (L, S, R) *be any global minimum vertex-cut in* G*. The running time of the algorithm is* $\tilde{O}\left(n^2\right)$*, and it is successful with respect to* (L, S, R) *with probability at least* $\frac{1}{12 \log n}$*.*

We apply Algorithm AlgSelectPairs′ from Claim 7.2 to graph G, and we let $\mathcal{E}$ be the good event that the algorithm is successful with respect to the distinguished cut (L, S, R). From Claim 7.2, $\mathbf{Pr}\left[\mathcal{E}\right] \geq \frac{1}{12 \log n}$.

If $\lambda > 2n^{1/2}$, then we terminate the algorithm and output an arbitrary vertex-cut. In this case, we say that the algorithm *terminates early*. Note that, if the distinguished cut is good, then we may only terminate the algorithm early in this step if Event $\mathcal{E}$ did not happen. From now on we assume that the algorithm did not terminate early. Let $\Gamma' \subseteq \Gamma$ be a collection of pairs of vertices obtained from Γ by discarding all pairs (x, y) where either $x = y$ or $(x, y) \in E(G)$ hold. Note that, if Event $\mathcal{E}$ happened, then there is a pair $(x^*, y^*) \in \Gamma$ of vertices with $x^* \in L$ and $y^* \in R$; in particular, $x^* \neq y^*$ and $(x^*, y^*) \notin E(G)$ must hold, so $(x^*, y^*) \in \Gamma'$ holds as well. Therefore, if $\Gamma' = \emptyset$, then it must be the case that Event $\mathcal{E}$ did not happen. If $\Gamma' = \emptyset$, we also terminate the algorithm and return an arbitrary vertex-cut; we say that the algorithm terminates early in this case as well. In the remainder of the algorithm, for brevity, we denote Γ' by Γ. For the sake of the analysis, we designate a single pair $(x^*, y^*) \in \Gamma$ of vertices as the *distinguished pair*, as follows. If Event $\mathcal{E}$ happened, then we let (x^*, y^*) be any pair of vertices with $x^* \in L$ and $y^* \in R$ (such a pair must exist from our discussion); in this case, we say that the distinguished pair is *good*. Otherwise, we let (x^*, y^*) be any pair of vertices in Γ, and we say that the distinguished pair is *bad*.

The main technical ingredient of our algorithm is a procedure that is summarized in the following theorem. Given a pair (x, y) of vertices in the graph G, the procedure computes a vertex-cut $(L_{x,y}, S_{x,y}, R_{x,y})$, so that, if Event $\mathcal{E}$ happened and (x, y) is the distinguished pair in Γ, then $(L_{x,y}, S_{x,y}, R_{x,y})$ is guaranteed to be a global minimum vertex-cut with a sufficiently high probability. The subroutine is given a query access to the (δ, τ)-subgraph oracle on the input graph G for appropriate values of δ and τ (see Definition 2.12).

THEOREM 7.3. *There is a randomized algorithm, whose input consists of a simple directed* 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, together with the adjacency-list representation of the split-graph* G' *of* G*. Additionally, the algorithm is given as input integers* $\lambda, \gamma \geq 1$ *with* $\lambda \cdot \gamma \leq \frac{n}{100}$*, and a pair* $x, y \in V(G)$ *of distinct vertices, such that* $(x, y) \notin E(G)$*. Lastly, the algorithm is given access to the* $(\delta, 20W_{\max}(G))$*-subgraph oracle for* G*, for* $\delta = 1 - \frac{\log(30000\lambda\gamma \cdot \log n)}{\log n}$*, so that* $n^{1-\delta} = 30000\lambda\gamma \cdot \log n$*. The algorithm returns an* x*-*y *vertex-cut* (L', S', R') *in* G *with the following guarantee: if there exists a minimum* x*-*y *vertex-cut* (L, S, R) *in* G *with* $|L| \leq \lambda$*, and if the subgraph oracle never erred in its responses to queries, then* (L', S', R') *is a minimum* x*-*y *vertex-cut in* G*. The expected running time of the algorithm is* $O\left(\left(\frac{n^{2+o(1)} \cdot \lambda}{\gamma} + n^{1+o(1)} \cdot \gamma \cdot \lambda\right) \cdot \log^5 W\right)$*, and it accesses the* $(\delta, 20W_{\max}(G))$*-subgraph oracle at most* $\log(80n(W+1))$ *times.*

We note that the algorithm from Theorem 7.3 may modify the adjacency-list representations of the graphs G and G' that it is given as input. We prove Theorem 7.3 in Subsection 7.3. At a high level, Theorem 7.3 can be viewed as an adaptation of Theorem 3.16 from [8] to directed graphs, that additionally obtains a better tradeoff between the running time and the parameter δ of the subgraph oracle. Additionally, the algorithm of [8] only works in the setting where there exists a minimum x-y vertex-cut (L, S, R) with $|L| \leq \frac{|S|}{n^{44/45}} \leq n^{1/45}$, while our algorithm works for much higher cardinalities of the set L, albeit with running time that depends linearly on $|L|$. While it is generally not difficult to extend the algorithm of [8] to the directed setting, we both significantly simplify the algorithm and its analysis, and obtain a faster running time, while also extending the algorithm to the setting where $|L|$ may be large. The latter is crucial in order to obtain the running time guarantees of Theorem 1.1. In the remainder of this subsection, we complete the proof of Theorem 7.1 using Theorem 7.3. This part of our algorithm follows a standard approach that is similar to that used by [8].

Completing the proof of Theorem 7.1. After computing the parameter λ and the set Γ of vertex pairs as described above, we compute the adjacency-list representations of the input graph G and its split-graph G'. Clearly, all these can be done in time $\tilde{O}(n^2)$. In the remainder of the description of the algorithm we assume that

it did not terminate early. The remainder of the algorithm consists of at most $z = \lfloor \log(80n(W+1)) \rfloor + 1$ phases.

We now fix an integer $1 \leq i \leq z$ and describe the execution of Phase i. At a very high level, our algorithm will apply the algorithm from Theorem 7.3 to every pair $(x,y) \in \Gamma$ of vertices one by one, while recording all random choices that the algorithm makes, until it performs the ith call to the subgraph oracle; we denote the corresponding oracle query by $Z_{i,(x,y)}$. Once it collects the ith query $Z_{i,(x,y)}$ for the subgraph oracle for all pairs $(x,y) \in \Gamma$, it calls to the algorithm from Lemma 2.13 in order to compute responses to all these queries at bulk. For every pair $(x,y) \in \Gamma$, when we execute the algorithm from Theorem 7.3 in the subsequent phase, we repeat all random choices and reuse all responses to oracle queries computed in previous phases (it is easy to verify that the first i oracle queries $Z_{1,(x,y)}, \ldots, Z_{i,(x,y)}$ that the algorithm asks will then remain unchanged). Therefore, when applying the algorithm from Theorem 7.3 to pair $(x,y) \in \Gamma$ in Phase i, we can assume that we have already computed the responses to the first $i-1$ queries to the subgraph oracle that the algorithm makes, and can therefore execute the algorithm until it makes its ith call to the oracle. We now describe the execution of the ith phase more formally.

We process every pair $(x,y) \in \Gamma$ of vertices one by one. When pair (x,y) is processed, we apply the algorithm from Theorem 7.3 to graph G, the the pair (x,y) of its vertices, the parameter λ computed above, and the parameter γ that is set to $\gamma = n^{(6-2\omega)/(11-3\omega)}$, where ω is the matrix multiplication exponent. Note that, using the current bounds $2 \leq \omega \leq 2.371552$ [37], $n^{1/4} \leq \gamma \leq n^{2/5}$ must hold. Moreover, since we assumed that the algorithm did not terminate early, $\lambda \leq 2n^{1/2}$, and, since we assumed that n is greater than a large enough constant, $\lambda \cdot \gamma \leq 2n^{9/10} \leq \frac{n}{100}$ must hold.

We execute the algorithm from Theorem 7.3 until its ith call to the subgraph oracle, while recording all random choices that it makes. Once the algorithm makes the ith call to the subgraph oracle, we terminate it, and we denote by $Z_{i,(x,y)}$ its ith query to the oracle. (If the algorithm terminates before making i queries to the subgraph oracle, then we instead let $Z_{i,(x,y)} = \emptyset$.) When the algorithm from Theorem 7.3 is subsequently executed with the pair (x,y) of vertices in Phase $(i+1)$, we repeat all the recorded random choices that we used when processing this pair in Phase i, so that the first i calls to the subgraph oracle are identical to the execution of the algorithm in Phase i. Lastly, once the algorithm from Theorem 7.3 terminates, we undo all the modifications that it made to the adjacency-list representations of G and G'. We then continue to process the next pair of vertices of Γ.

Once all pairs of vertices in Γ are processed in this manner, we execute the algorithm from Lemma 2.13 with parameters $\delta = 1 - \frac{\log(30000\lambda\gamma \cdot \log n)}{\log n}$ and $\tau = 20W_{\max}(G)$ in order to compute the responses to all queries $\left\{Z_{i,(x,y)} \mid (x,y) \in \Gamma\right\}$ to the subgraph oracle. Observe that, since we assumed that the algorithm did not terminate early, $\lambda \leq 2n^{1/2}$ must hold. Since, as observed already, $n^{1/4} \leq \gamma \leq n^{2/5}$, and since we assumed that n is greater than a large enough constant, we get that $\lambda \cdot \gamma \leq 2n^{9/10} < \frac{n}{30000 \cdot 2^{\sqrt{\log n}} \cdot \log n}$ and $\log(30000\lambda\gamma \cdot \log n) < \log n - \sqrt{\log n}$ must hold. Therefore, $\delta = 1 - \frac{\log(30000\lambda\gamma \cdot \log n)}{\log n} > \frac{1}{\sqrt{\log n}}$. Additionally, since, as observed already, $\gamma \geq n^{1/4}$, we get that $\delta \leq \frac{3}{4}$ must hold as well. We conclude that we obtain a valid input to the algorithm from Lemma 2.13. Once the algorithm from Lemma 2.13 computes the responses to all queries $\left\{Z_{i,(x,y)} \mid (x,y) \in \Gamma\right\}$, the current phase terminates and we continue to Phase $(i+1)$. When processing the vertex pair $(x,y) \in \Gamma$ in Phase $(i+1)$, we repeat all random choices made in Phase i. Note that at this time we have already computed responses to the first i queries of this algorithm to the subgraph oracle, so we can now execute this algorithm until its $(i+1)$th query to the oracle.

Finally, for every pair $(x,y) \in \Gamma$ of vertices, we denote by $(L_{x,y}, S_{x,y}, R_{x,y})$ the vertex-cut that the algorithm from Theorem 7.3 returned when applied to the pair (x,y) in the last phase. We let $(x',y') \in \Gamma$ be the pair for which $w(S_{x',y'})$ is minimized, and we return the corresponding vertex-cut $(L_{x',y'}, S_{x',y'}, R_{x',y'})$. This concludes the description of the algorithm. We now proceed to analyze it.

Let $\mathcal{E}^{\text{oracle}}$ be the bad event that the algorithm from Lemma 2.13 errs in its response to any query in any of the phases. Recall that the number of phases in the algorithm is bounded by:

$$z = \lfloor \log(80n(W+1)) \rfloor + 1 \leq 4\log(80nW).$$

Recall also that we apply the subgraph oracle with the parameter $\tau = 20W_{\max}(G) \geq 20nW'_{\max}(G) \geq 8nW$

(from the definitions of $W_{\max}(G)$ and $W'_{\max}(G)$ in Subsection 2.4, and from our assumption that $W = \max_{v \in V(G)} \{w(v)\}$).

By combining Lemma 2.13 with the Union Bound, we get that:

$$\mathbf{Pr}\left[\mathcal{E}^{\text{oracle}}\right] \leq \frac{z}{n^9 \cdot \log^4 \tau} \leq \frac{4\log(80nW)}{n^9 \cdot \log^4(8nW)} \leq \frac{1}{64n}.$$

Let $\mathcal{E}^*$ be the good event that Event $\mathcal{E}$ happened and Event $\mathcal{E}^{\text{oracle}}$ did not happen. Recall that, from Claim 7.2, $\mathbf{Pr}\left[\mathcal{E}\right] \geq \frac{1}{12\log n}$, and from our discussion above, $\mathbf{Pr}\left[\mathcal{E}^{\text{oracle}}\right] \leq \frac{1}{64n}$. Altogether, we get that:

$$\mathbf{Pr}\left[\neg\mathcal{E}^*\right] \leq \mathbf{Pr}\left[\neg\mathcal{E}\right] + \mathbf{Pr}\left[\mathcal{E}^{\text{oracle}}\right] \leq 1 - \frac{1}{12\log n} + \frac{1}{64n} \leq 1 - \frac{1}{24\log n},$$

and so $\mathbf{Pr}\left[\mathcal{E}^*\right] \geq \frac{1}{24\log n}$. We use the following observation to complete the correctness analysis of the algorithm.

OBSERVATION 7.4. *Assume that the distinguished cut is good and that Event $\mathcal{E}^*$ has happened. Then the cut that the algorithm outputs is a global minimum vertex-cut.*

Proof. Recall that if the distinguished cut is good and Event $\mathcal{E}$ has happened, then the algorithm may not terminate early, and moreover, the distinguished pair $(x^*, y^*) \in \Gamma$ of vertices is good; in other words, $x^* \in L$ and $y^* \in R$ holds. Therefore, the distinguished cut (L, S, R) is also a minimum x^*-y^* vertex-cut in G, and, since the distinguished cut is good, $|L| \leq \lambda$ holds. Since we assumed that Event $\mathcal{E}^{\text{oracle}}$ did not happen, when the algorithm from Theorem 7.3 is applied to the pair (x^*, y^*) in the last phase, the cut $(L_{x^*,y^*}, S_{x^*,y^*}, R_{x^*,y^*})$ that it returns must be a minimum x^*-y^* vertex-cut in G, so its value is $w(S_{x^*,y^*}) = w(S)$. Since our algorithm returns a smallest-value vertex-cut from among the cuts in $\{(L_{x,y}, S_{x,y}, R_{x,y}) \mid (x,y) \in \Gamma\}$, the value of the cut that it returns must be $w(S)$, and so it must be a global minimum vertex-cut. ∎

We conclude that, with probability at least $\frac{1}{24\log n}$, the vertex-cut that the algorithm returns is a global minimum vertex-cut.

It now remains to bound the running time of the algorithm. As noted already, the time required to execute Algorithm AlgSelectPairs′, and to compute the adjacency-list representations of the graphs G and G' is bounded by $\tilde{O}(n^2)$. The number of phases in the remainder of the algorithm is $z \leq O(\log n \log W)$. In every phase, we perform $|\Gamma| \leq \tilde{O}\left(\frac{n}{\lambda}\right)$ iterations, each of which executes the algorithm from Theorem 7.3, whose expected running time is $O\left(\left(\frac{n^{2+o(1)} \cdot \lambda}{\gamma} + n^{1+o(1)} \cdot \gamma \cdot \lambda\right) \cdot \log^5 W\right)$. Therefore, the total expected running time of all applications of the algorithm from Theorem 7.3 is bounded by:

$$\begin{aligned}
&O\left(\left(\frac{n^{2+o(1)} \cdot \lambda}{\gamma} + n^{1+o(1)} \cdot \gamma \cdot \lambda\right) \cdot z \cdot |\Gamma| \cdot (\log W)^{O(1)}\right) \\
&\quad \leq O\left(\left(\frac{n^3}{\gamma} + n^2 \cdot \gamma\right) \cdot n^{o(1)} \cdot (\log W)^{O(1)}\right) \\
&\quad \leq O\left(\left(n^{2+(5-\omega)/(11-3\omega)} + n^{2+(6-2\omega)/(11-3\omega)}\right) \cdot n^{o(1)} \cdot (\log W)^{O(1)}\right) \\
&\quad \leq O\left(\left(n^{2+(5-\omega)/(11-3\omega)+o(1)}\right) \cdot (\log W)^{O(1)}\right).
\end{aligned}$$

(We have used the fact that $\gamma = n^{(6-2\omega)/(11-3\omega)} = n^{1-(5-\omega)/(11-3\omega)}$).

Lastly, in every phase we also execute the algorithm from Lemma 2.13, whose running time is bounded in the following simple but technical claim, whose proof is deferred to the full version of the paper.

CLAIM 7.5. *The total running time that the algorithm spends, over all phases, on the algorithm from Lemma* 2.13 *is bounded by $O\left(n^{2+(5-\omega)/(11-3\omega)+o(1)} \cdot (\log W)^{O(1)}\right)$.*

Overall, the expected running time of the entire algorithm is bounded by

$$O\left(n^{2+(5-\omega)/(11-3\omega)+o(1)} \cdot (\log W)^{O(1)}\right). \tag{7.1}$$

In order to achieve the worst-case running time bound stated in Theorem 7.1, we terminate the algorithm whenever its running time exceeds the expected running time by a factor of $48 \log n$, in which case we output an arbitrary vertex-cut in G. By Markov's inequality, the probability that the algorithm is terminated for this reason is at most $\frac{1}{48 \log n}$. Overall, if the distinguished cut is good, then the probability that algorithm outputs a global minimum vertex-cut remains at least $\frac{1}{24 \log n} - \frac{1}{48 \log n} \geq \frac{1}{48 \log n}$. In order to complete the proof of Theorem 7.1, it is now enough to prove Theorem 7.3, which we do next.

7.3 The Main Subroutine: Proof of Theorem 7.3 In this section we provide the algorithm for our main subroutine, proving Theorem 7.3. A key notion used in our proof is that of well-behaving cuts, that we define next.

DEFINITION 7.6 (A well-behaving cut). *Let G be a directed n-vertex graph, and let $x, y \in V(G)$ be a pair of its distinct vertices. Let $\lambda, \gamma \geq 1$ be integral parameters, and let $M' > 0$ be another parameter. Given a pair (L, S, R) and (L', S', R') of x-y vertex-cuts in G, we say that (L', S', R') is* (λ, γ)-well-behaving for scale M' with respect to (L, S, R), *if all of the following hold:*

W1. $\mathrm{Vol}^+_G(L') \leq \frac{100n^2}{\gamma}$;

W2. $w(S') \leq w(S) + 10nM'$; *and*

W3. set $(L \cup S) \setminus (L' \cup S')$ contains at most $20\lambda\gamma$ vertices of weight at least M'.

Note that in the above definition it is possible that $M' < 1$. Note also that the notion of a well-behaving cut is somewhat similar to the notion of vertex-AFMC (see Definition 4.5). The main difference is the specific setting of the parameters; that Property W3 only extends to vertices of weight at least M' and is only required to hold for a specific vertex-cut (L, S, R); and the additional Requirement W1 that L' has a small out-volume.

The Distinguished Cut. For the sake of the analysis, we fix a minimum x-y vertex-cut (L, S, R), that we refer to as the *distinguished cut*, as follows: if there exists a minimum x-y vertex-cut $(\hat{L}, \hat{S}, \hat{R})$ with $|\hat{L}| \leq \lambda$, then let (L, S, R) be any such vertex-cut, and we say that the distinguished cut is *good*. Otherwise, we let (L, S, R) be any minimum x-y vertex-cut, and we say that the distinguished cut is *bad*.

The main idea of our algorithm is to iteratively construct x-y vertex-cuts that are well-behaving with respect to the distinguished cut for increasingly smaller scales.

For an integer $i \geq 1$, let $M_i = \frac{W'_{\max}(G)}{2^i}$; observe that M_i is an integral power of 2. Our algorithm consists of $z = \lceil \log(20nW'_{\max}(G)) \rceil$ phases. Intuitively, the ith phase will focus on obtaining an x-y vertex-cut that is well-behaving with respect to the distinguished cut for the scale M_i

Specifically, for all $1 \leq i \leq z$, the input to Phase i is an x-y vertex-cut $(L_{i-1}, S_{i-1}, R_{i-1})$ in G that has the following property: if the distinguished cut (L, S, R) is good, then the cut $(L_{i-1}, S_{i-1}, R_{i-1})$ is (λ, γ)-well-behaving for scale M_{i-1} with respect to (L, S, R). The output of Phase i is an x-y vertex-cut (L_i, S_i, R_i), that can serve as a valid input to Phase $(i+1)$. In other words, it must have the following property: if the distinguished cut (L, S, R) is good, then (L_i, S_i, R_i) is (λ, γ)-well-behaving for scale M_{i-1} with respect to (L, S, R). The algorithm for a single phase is summarized in the following theorem, whose proof is deferred to the full version of the paper.

THEOREM 7.7. *There is a randomized algorithm, whose input consists of a simple directed 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, together with the adjacency-list representation of the split-graph G' of G. Additionally, the algorithm is given as input integral parameters $\lambda, \gamma \geq 1$ with $\lambda \cdot \gamma \leq \frac{n}{100}$, a pair $x, y \in V(G)$ of distinct vertices such that $(x, y) \notin E(G)$, a scale parameter $\frac{1}{100n} \leq M' \leq \frac{W'_{\max}(G)}{2}$ that is an integral power of 2, and an x-y vertex-cut (L', S', R') in G. Lastly, the algorithm is allowed to make a single query to the $(\delta, 20W_{\max}(G))$-subgraph oracle for G, for*

$\delta = 1 - \frac{\log(30000\lambda\gamma \cdot \log n)}{\log n}$, *so that* $n^{1-\delta} = 30000\lambda\gamma \cdot \log n$. *The algorithm returns an* x-y *vertex-cut* (L'', S'', R'') *with the following guarantee: if there exists a minimum* x-y *vertex-cut* (L, S, R) *with* $|L| \leq \lambda$ *such that the cut* (L', S', R') *is* (λ, γ)*-well-behaving for scale* $(2M')$ *with respect to* (L, S, R)*, and if the subgraph oracle did not err in its response to the query, then cut* (L'', S'', R'') *is* (λ, γ)*-well-behaving for scale* M' *with respect to* (L, S, R)*. The expected running time of the algorithm is* $O\left(\left(\frac{n^{2+o(1)} \cdot \lambda}{\gamma} + n^{1+o(1)} \cdot \gamma \cdot \lambda\right) \cdot \log^4 W\right)$.

We note that the algorithm from Theorem 7.7 may modify the adjacency-list representations of the graphs G and G' that it is given as input. We are now ready to complete the proof of Theorem 7.3. Our algorithm starts by computing an initial x-y vertex-cut (L_0, S_0, R_0) with $L_0 = \{x\}$, $S_0 = N^+_G(x)$, and $R_0 = V(G) \setminus (L_0 \cup S_0)$; we later show that, if the distinguished cut is good, then (L_0, S_0, R_0) is (λ, γ)-well-behaving for scale M_0 with respect to it. We then perform $z = \lceil \log(20nW'_{\max}(G)) \rceil$ phases. For all $1 \leq i \leq z$, we use the algorithm from Theorem 7.7 with scale parameter $M' = M_i$ and cut $(L', S', R') = (L_{i-1}, S_{i-1}, R_{i-1})$. We denote the cut that the algorithm from Theorem 7.7 outputs by (L_i, S_i, R_i). Recall that the algorithm from Theorem 7.7 may modify the adjacency-list representations of the graphs G and G' that it is given as input; at the end of every phase we undo all these changes, so that the algorithm for the subsequent phase is given access to a valid adjacency-list representations of the graphs G and G'. After the completion of the last phase, the algorithm returns the cut (L_z, S_z, R_z) computed in that phase. This completes the description of the algorithm. We now proceed to analyze its correctness, starting with the following simple claim.

Claim 7.8. *Cut* (L_0, S_0, R_0) *is a valid* x-y *vertex-cut in* G*, and, moreover, it is* (λ, γ)*-well-behaving for scale* M_0 *with respect to the distinguished cut* (L, S, R).

Proof. It is immediate to verify that (L_0, S_0, R_0) is a valid x-y vertex-cut from its definition, and from the fact that $(x, y) \notin E(G)$.

It now remains to prove that (L_0, S_0, R_0) is (λ, γ)-well-behaving for scale M_0 with respect to the distinguished cut (L, S, R). First, since $|L_0| = 1$, and $\gamma \leq \lambda \cdot \gamma \leq n$, we get that $\mathrm{Vol}^+_G(L_0) \leq n \leq \frac{100n^2}{\gamma}$, establishing Property W1. Next, recall that $M_0 = W'_{\max}(G) > \max_{v \in G}\{w(v)\}$, so $w(S_0) \leq w(V(G)) \leq n \cdot W'_{\max}(G) \leq 10nM_0$ holds, establishing Property W2. Lastly, since the weight of every vertex in G is less than M_0, we get that the set $(L \cup S) \setminus (L_0 \cup S_0)$ contains no vertices with weight at least M_0, establishing Property W3. ∎

We then obtain the following immediate corollary.

Corollary 7.9. *For all* $0 \leq i \leq z$*,* (L_i, S_i, R_i) *is a valid* x-y *vertex-cut in* G*. Moreover, if the distinguished cut* (L, S, R) *is good, and if the subgraph oracle never erred in its responses to its queries, then the vertex-cut* (L_z, S_z, R_z) *is* (λ, γ)*-well-behaving for scale* M_z *with respect to* (L, S, R).

The proof of the corollary follows immediately by induction on i, where the base of the induction, with $i = 0$, follows from Claim 7.8, and the step follows from the statement of Theorem 7.7.

It remains to show that, if the distinguished cut is good, and if the subgraph oracle never erred in its responses to the queries, then the x-y vertex-cut (L_z, S_z, R_z) returned by our algorithm is a minimum x-y vertex-cut. Indeed, assume that the distinguished cut (L, S, R) is good, and that the subgraph oracle never erred in its responses to the queries. Then, from Corollary 7.9, the vertex-cut (L_z, S_z, R_z) is (λ, γ)-well-behaving for scale M_z with respect to (L, S, R). By property W2 of well-behaving cuts, $w(S_z) \leq w(S) + 10nM_z < w(S) + 1$ holds, since, from the definition of the parameter z, $M_z \leq \frac{1}{20n}$. Since all vertex weights are integral, it must be the case that $w(S_z) \leq w(S)$, and so (L_z, S_z, R_z) is a minimum x-y vertex-cut.

Running Time and Oracle Access. The time required to compute the initial vertex-cut (L_0, S_0, R_0) is $O(n)$. Since the number of phases is $z \leq \lceil \log(20nW'_{\max}(G)) \rceil \leq O(\log(nW)) \leq O(\log n \cdot \log W)$, and the expected running time of every phase, from Theorem 7.7, is $O\left(\left(\frac{n^{2+o(1)} \cdot \lambda}{\gamma} + n^{1+o(1)} \cdot \gamma \cdot \lambda\right) \cdot \log^4(W)\right)$, we get that the total expected running time of the algorithm is:

$$O\left(n + \left(\frac{n^{2+o(1)} \cdot \lambda}{\gamma} + n^{1+o(1)}\gamma\lambda\right) \cdot \log^5(W)\right) \leq O\left(\left(\frac{n^{2+o(1)} \cdot \lambda}{\gamma} + n^{1+o(1)} \cdot \gamma \cdot \lambda\right) \cdot \log^5(W)\right).$$

Lastly, observe that the algorithm accesses the $(\delta, 20W_{\max}(G))$-subgraph oracle at most once per phase, so the total number of times it accesses the oracle is bounded by $\lceil \log(20nW'_{\max}(G)) \rceil \leq \log(40nW'_{\max}(G)) \leq \log(80n(W+1))$.

References

[1] 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, 2021, pp. 1725–1737, https://doi.org/10.1145/3406325.3451073.

[2] A. Anand, T. Saranurak, and Y. Wang, *Deterministic edge connectivity and max flow using subquadratic cut queries*, in Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2025, pp. 124–142.

[3] S. Baswana, K. Bhanja, and A. Roy, *Faster algorithm for second (s, t)-mincut and breaking quadratic barrier for dual edge sensitivity for (s, t)-mincut*, arXiv preprint arXiv:2507.01366, (2025).

[4] 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, 2021, pp. 1147–1158, https://doi.org/10.1109/FOCS52979.2021.00113.

[5] 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, 2017, pp. 1900–1918, https://doi.org/10.1137/1.9781611974782.124.

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

[7] 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), 2022, pp. 612–623, https://doi.org/10.1109/FOCS54457.2022.00064.

[8] J. Chuzhoy and O. Trabelsi, *Breaking the $O(mn)$-time barrier for vertex-weighted global minimum cut*, in Proceedings of the 57th Annual ACM Symposium on Theory of Computing, 2025, pp. 144–155.

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

[10] O. Fischer, Y. Jiang, S. Mukhopadhyay, and S. Yingchareonthawornchai, *Directed and undirected vertex connectivity problems are equivalent for dense graphs*. SOSA 2026, to appear. A preliminary version of the paper can be found at https://arxiv.org/abs/2508.20305.

[11] 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 Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, S. Chawla, ed., 2020, pp. 2046–2065, https://doi.org/10.1137/1.9781611975994.126.

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

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

[14] J. Hao and J. B. Orlin, *A faster algorithm for finding the minimum cut in a directed graph*, J. Algorithms, 17 (1994), pp. 424–446, https://doi.org/10.1006/JAGM.1994.1043. Announced at SODA 1992.

[15] 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, 2024, pp. 3089–3139, https://doi.org/10.1137/1.9781611977912.111.

[16] M. Henzinger, S. Rao, and D. Wang, *Local flow partitioning for faster edge connectivity*, in Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, P. N. Klein, ed., 2017, pp. 1919–1938, https://doi.org/10.1137/1.9781611974782.125.

[17] M. R. Henzinger, S. Rao, and H. N. Gabow, *Computing vertex connectivity: New bounds from old techniques*, J. Algorithms, 34 (2000), pp. 222–250, https://doi.org/10.1006/JAGM.1999.1055. Announced at FOCS 1996.

[18] Y. Jiang, C. Nalam, T. Saranurak, and S. Yingchareonthawornchai, *Deterministic vertex connectivity via common-neighborhood clustering and pseudorandomness*, in Proceedings of the 57th Annual ACM Symposium on Theory of Computing, 2025, pp. 2293–2304.

[19] D. Kang and J. Payor, *Flow rounding*, CoRR, (2015), https://arxiv.org/pdf/1507.08139.

[20] D. R. Karger, *Minimum cuts in near-linear time*, J. ACM, 47 (2000), pp. 46–76, https://doi.org/10.1145/331605.331608. Announced at STOC 1996.

[21] D. R. Karger and C. Stein, *An $\tilde{O}(n^2)$ algorithm for minimum cuts*, in Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993, S. R. Kosaraju, D. S. Johnson, and A. Aggarwal, eds., 1993, pp. 757–765, https://doi.org/10.1145/167088.167281.

[22] K. Kawarabayashi and M. Thorup, *Deterministic edge connectivity in near-linear time*, J. ACM, 66 (2019), pp. 4:1–4:50, https://doi.org/10.1145/3274663.

[23] D. J. Kleitman, *Methods for investigating connectivity of large graphs*, IEEE Transactions on Circuit Theory, 16 (1969), pp. 232–233, https://doi.org/10.1109/TCT.1969.1082941.

[24] J. Li, *Deterministic mincut in almost-linear time*, in Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021, S. Khuller and V. V. Williams, eds., 2021, pp. 384–395, https://doi.org/10.1145/3406325.3451114.

[25] 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, 2021, pp. 317–329, https://doi.org/10.1145/3406325.3451088.

[26] 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, 2020, pp. 85–92, https://doi.org/10.1109/FOCS46700.2020.00017.

[27] N. Linial, L. Lovász, and A. Wigderson, *Rubber bands, convex embeddings and graph connectivity*, Combinatorica, 8 (1988), pp. 91–102, https://doi.org/10.1007/BF02122557.

[28] Y. Mansour and B. Schieber, *Finding the edge connectivity of directed graphs*, J. Algorithms, 10 (1989), pp. 76–85, https://doi.org/10.1016/0196-6774(89)90024-2.

[29] 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, 2019, pp. 241–252, https://doi.org/10.1145/3313276.3316394.

[30] C. S. A. Nash-Williams, *Edge-disjoint spanning trees of finite graphs*, Journal of the London Mathematical Society, s1-36 (1961), pp. 445–450, https://doi.org/10.1112/jlms/s1-36.1.445.

[31] V. D. Podderyugin, *An algorithm for finding the edge connectity of graphs*, 1973, https://cir.nii.ac.jp/crid/1371694371159352077.

[32] C. Schnorr, *Bottlenecks and edge connectivity in unsymmetrical networks*, SIAM J. Comput., 8 (1979), pp. 265–274, https://doi.org/10.1137/0208019.

[33] D. D. Sleator and R. E. Tarjan, *A data structure for dynamic trees*, J. Comput. Syst. Sci., 26 (1983), pp. 362–391, https://doi.org/10.1016/0022-0000(83)90006-5.

[34] O. Trabeisi, *(almost) ruling out seth lower bounds for all-pairs max-flow*, in Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2025, pp. 2132–2156.

[35] W. T. Tutte, *On the problem of decomposing a graph into n connected factors*, Journal of the London Mathematical Society, s1-36 (1961), pp. 221–230, https://doi.org/10.1112/jlms/s1-36.1.221.

[36] 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, 2023, pp. 503–514, https://doi.org/10.1109/FOCS57990.2023.00037.

[37] V. V. Williams, Y. Xu, Z. Xu, and R. Zhou, *New bounds for matrix multiplication: from alpha to omega*, in Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, D. P. Woodruff, ed., 2024, pp. 3792–3835, https://doi.org/10.1137/1.9781611977912.134.