Local Flow Partitioning for Faster Edge Connectivity

Monika Henzinger [*] Satish Rao[†] Di Wang [‡]

Abstract

We study the problem of computing a minimum cut in a simple, undirected graph and give a deterministic $O(m \log^2 n \log\log^2 n)$ time algorithm. This improves both on the best previously known deterministic running time of $O(m \log^{12} n)$ (Kawarabayashi and Thorup [12]) and the best previously known randomized running time of $O(m \log^3 n)$ (Karger [11]) for this problem, though Karger's algorithm can be further applied to weighted graphs.

Our approach is using the Kawarabayashi and Thorup graph compression technique, which repeatedly finds low-conductance cuts. To find these cuts they use a diffusion-based local algorithm. We use instead a flow-based local algorithm and suitably adjust their framework to work with our flow-based subroutine. Both flow and diffusion based methods have a long history of being applied to finding low conductance cuts. Diffusion algorithms have several variants that are naturally local while it is more complicated to make flow methods local. Some prior work has proven nice properties for local flow based algorithms with respect to improving or cleaning up low conductance cuts. Our flow subroutine, however, is the first that is both local and produces low conductance cuts. Thus, it may be of independent interest.

1 Introduction

Given an unweighted (or simple) graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, the edge connectivity λ of G is the size of the smallest edge set whose removal disconnects the graph. Given an edge-weighted graph $G_w = (V, E, w)$ the minimum cut of G_w is the weight of the minimum weight edge set whose removal disconnects the graph. In a breakthrough paper in 1996, Karger [11] gave the first randomized algorithm that computes the minimum cut in expected near-linear time and posed as an open question to find a deterministic near-linear time minimum cut algorithm. Almost 20 years later, in a recent breakthrough, Kawarabayashi and Thorup [12] partially answered his open question, by presenting the first deterministic near-linear time algorithm for finding edge connectivity in an unweighted simple graph. They state their runtime as $O(m \log^{12} n)$.

Their contribution is on two levels. They improved the deterministic runtime for edge connectivity to near linear time, and perhaps of more or equal interest they developed a new deterministic algorithm that computes from G a sparser multi-graph $\overline{G}$ that preserves all non-trivial minimum cuts in G, i.e., a deterministic sparsification of G. We note that $\overline{G}$ is produced by a recursive procedure and we refer to either $\overline{G}$ or the procedure as the *K-T decomposition.* Applying Gabow's edge connectivity algorithm [7] (which runs on multi-graphs) to $\overline{G}$ yields the claimed results for computing edge connectivity.

We believe the K-T decomposition is of independent interest based on the long line of research on sparsification and clustering and the astounding impact in algorithms sparsification and clustering have had. For example, the near-linear time solvers for linear systems [20, 5, 13] is one of the very important applications of sparsification. This specific application as well as a large part of the prior work on sparsification is based on randomization. As the K-T decomposition is deterministic and it introduces some quite interesting ideas and structures, we believe it might well prove useful in improving the state of the art with respect to the co-evolution of graph decompositions and algorithms, and specifically the use of deterministic sparsification in various algorithms.

A central tool used in [12] to compute the K-T decomposition is a local *probability mass diffusion method*, called a "page rank" method. We replace this diffusion method by a flow-based method and modify the K-T algorithm to accomodate the differences between these methods. As a result we derive an algorithm that has a deterministic runtime of $O(m \log^2 n \log\log^2 n)$ for com-

[*] University of Vienna, monika.henzinger@univie.ac.at. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506 and was done in part while Monika Henzinger was visiting the Simons Institute of the Theory of Computing.

[†] UC Berkeley, satishr@cs.berkeley.edu. The author was supported by NSF Grant CCF-1528174 and CCF-1535989 during this work.

[‡] UC Berkeley, wangd@eecs.berkeley.edu. The author was supported by NSF Grant CCF-1528174 and CCF-1535989 during this work.

puting a K-T decomposition and the edge connectivity in G. Note that our deterministic algorithm is faster than the best known randomized edge connectivity algorithm, whose running time is $O(m \log^3 n)$ [11].

Flows versus diffusion methods. From a technical point of view our result contributes to the line of work on finding low conductance cuts using local methods; a *local method* being one whose runtime depends only on the volume of the (smaller side of the) cut that it outputs. Flow and probability mass diffusions (or more generally, spectral methods) have a long history of competing to provide good graph decompositions. But diffusions have the upper hand in terms of local methods, as the fact that the diffusion process is a linear operator allows for the detailed knowledge of its evolution, which then can be used to reason powerfully about its behavior. For example, Spielman and Teng [20], inspired by classical analyses of random walks by Lovász and Simonovitz [15], showed that most vertices in a low conductance cut are good starting points for a diffusion that finds a good cut. Past flow based methods, however, were subject to the black-box of a flow algorithm that could adversarly send flow in an inopportune direction. We overcome this difficulty by suitably modifying the flow computation and present the first primarily flow-based local method for locating low conductance cuts. We expect that our approach can used to speed up further conductance-based graph algorithms.

Previously, the methods of [18, 14] combined the properties of diffusions and flow algorithms to produce low conductance cuts. The methods alternate between diffusions which find barely non-trivial cuts in an embedded graph, and flow embedding edges to cross these cuts. The flow computation interacts with the original graph without the quadratic (in conductance) loss that is inherent in diffusion methods[1]. Those methods, however, fail terribly to give local methods for finding low conductance cuts, and explictly treat the flow algorithm more as an constrained adversary rather a useful tool.

Our flow algorithm attempts to combine (some of) the power of diffusion with the speed and efficiency of flow methods more tightly, without actually using a diffusion method. Basically it consists of an excess scaling method repeatedly calling a modified push-relabel algorithm. The excess scaling portion enforces locality on our algorithm and the details of our push-relabel method allow us to get a handle on conductance.

In recent work, some local flow based methods have been studied in a similar vein. However, diffusions are still used when producing low conductance cuts. For example, Orecchia and Zhu [19] use a detailed view of a blocking flow based flow algorithm to obtain improved results on finding low conductance cuts; in particular, they show how to locally find a $\tilde{O}(\frac{1}{\gamma})$ approximation to conductance given a seed set overlapping the cut by a γ fraction. They apply their method to local partitioning but use a result of Allen, Lattanzi and Mirrokni [22] which in turn uses diffusion or page-rank from [1] (and as does [12]).

In the context of [12], our flow method roughly matches diffusion where it does well, and dominates diffusion with respect to its quadratic loss. That is, the decomposition developed in [12], repeatedly finds cuts of low conductance $O(\frac{1}{\log^c n})$, or certifies a certain property related to connectivity. In [12], the local diffusions suffer both a quadratic gap (as well as a logarithmic factor) between what can be certified and the conductance of a cut as well as quadratic (in the conductance) overhead in runtime. Our modified push-relabel algorithm either certifies the property or finds a low conductance cut with only a logarithmic gap. However, its runtime depends on the amount of "source supply" provided to it. To make sure that this "source supply" is only $O(m \log n)$ we use the excess scaling procedure, which repeatedly calls the push-relabel algorithm with suitably rescaled source supply. This leads to the improvement in runtime for our method.

Other Previous Work. Work on edge connectivity and its generalization, the minimum cut problem, has a long history perhaps beginning with Gomory and Hu's [9] use of the maximum flow problem to solve this problem. Some relatively recent highlights include the work of Nagamochi and Ibaraki [17] which bypasses the use of the maximum flow problem, and simple beautiful versions of these by Frank [6] and Stoer and Wagner [21] which give $O(nm+n^2 \log n)$ deterministic algorithms for minimum cut.

For edge connectivity of simple graphs, Gabow had the best previous deterministic algorithm which was $O(m + \lambda^2 n)$ time where λ was the connectivity. His methods could handle parallel edges in $O(m+\lambda^2 n \log n)$ time. Matula [16] has a linear time $(2+\epsilon)$ approximation algorithm for this problem as well.

There is also substantial work in local graph partitioning including the aforementioned work of Anderson, Chung, and Lang [1] which gives a local diffusion process that outputs a set of conductance $(\phi \log n)^{-1/2}$ in time $O(\phi^{-1} \log^c n)$ times the size of the output for a good fraction of the starting vertices in a cut of conductance ϕ. The runtime overhead was improved to $\phi^{-1/2}$ using an evolving set diffusion by Anderson and Peres [2]. The heat kernel diffusion was used to improve the quality of the cut to $\phi^{-1/2}$ in [4], though the impact on

[1] Indeed, one could also see the best known approximation algorithm for conductance in [3] as such a combination.

runtime overhead is not clear in that work. We note that the result of local diffusions have also had impact empirically in, for example, the use of Personalized Page Pank [10].

Organization of Abstract. As our algorithm is quite involved and builds upon the K-T framework, we only have space to present our main contribution, namely the flow algorithm in some detail. For the other parts of our algorithm we can, for space reasons, only present the main idea and the intuition in the extended abstract. Specifically, we present our flow procedure along with its properties in Section 3, and include the proofs in Appendix B. We describe the overall structure of the K-T decomposition in Section 4, with some details deferred to Appendix A. We then present our version of the K-T inner procedure in Section 5, and a detailed analysis in Appendix C. Finally, Section 6 contains the runtime analysis.

2 Preliminaries and notations

We denote $d(v)$ as the degree v, and $\mathbf{vol}(C)$ as the *volume* of $C \subseteq V$. The *internal edges* of a set $C \subseteq V$ are the edges with both endpoints in C. We add H or C as subscripts, i.e. $d_H(v), \mathbf{vol}_C(A)$ etc., if we consider only the internal edges of a subgraph H or a subset $C \subseteq V$, while we omit the subscripts when the graph is clear from context. We use m to denote the number of (internal) edges of a graph, and again add subscript to m when there are multiple graphs in the context.

A *cut* is a subset $S \subset V$, or $(S, \bar{S})$ where $\bar{S} = V \setminus S$. The *cut-size* $\partial(S)$ of a cut S is the number of edges between S and $\bar{S}$. A cut S is *non-trivial* if $|S|, |\bar{S}| > 1$. The *conductance* of a cut S is $\Phi(S) \overset{\text{def}}{=} \frac{\partial(S)}{\min(\mathbf{vol}(S), \mathbf{vol}(V \setminus S))}$. Unless otherwise noted, when speaking of the conductance of a cut S, we assume S to be the side of minimum volume. Given $A \subset V$ and a cut S we say that A *contains* the cut S if there exist nodes u and v in A such that $u \in S$ and $v \in \bar{S}$. Otherwise, we say that A *does not contain* the cut.

We will consider flow problems extensively. Formally, a *flow problem* is defined with a *source function*, $\Delta : V \to \mathbb{Z}_{\geq 0}$, a *sink function*, $T : V \to \mathbb{Z}_{\geq 0}$, and edge capacities $c(\cdot)$. We say that v is *a sink of capacity* x if $T(v) = x$. All flow problems we consider in this work use the same sink function, $\forall v : T(v) = d(v)$, so we won't explicitly write down $T(\cdot)$. To avoid confusion with the way flow is used, we use *supply* to refer to the substance being routed in flow problems.

For the sake of efficiency, we will not typically obtain a full solution to a flow problem. We will compute a pre-flow, which is a function $f : V \times V \to R$, where $f(u, v) = -f(v, u)$. A pre-flow f is *source-feasible* with respect to source function Δ if $\forall v : \sum_u f(v, u) \leq \Delta(v)$. A pre-flow f is *capacity-feasible* with respect to $c(\cdot)$ if $|f(u, v)| \leq c(e)$ for $e = \{u, v\} \in E$ and $f(u, v) = 0$ otherwise. We say that f is a *feasible pre-flow* for flow problem Π, or simply a *pre-flow* for Π, if f is both source-feasible and capacity-feasible with respect to Π.

For a pre-flow f and a source function $\Delta(\cdot)$, we extend the notation to denote $f(v) \overset{\text{def}}{=} \Delta(v) + \sum_u f(u, v)$ as *the amount of supply ending at* v *after* f. Note that $f(v)$ is non-negative for all v if f is source-feasible. When we use a pre-flow as a function on vertices, we refer to the function $f(\cdot)$, and it will be clear from the context what $\Delta(\cdot)$ we are using. If in addition, $\forall v : f(v) \leq T(v)$, the pre-flow f will be a *feasible flow (solution)* to the flow problem Π.

We denote $\mathrm{ex}(v) \overset{\text{def}}{=} \max(f(v) - T(v), 0)$ as the excess supply at v, and we call the part of the supply below $T(v)$ as the supply routed to the sink at v, or *absorbed* by v. We call the sum of all the supply absorbed by vertices, $\sum_v \min(f(v), T(v))$, the total supply routed to sinks. Finally, given a source function $\Delta(\cdot)$, we define $|\Delta(\cdot)| \overset{\text{def}}{=} \sum_v \Delta(v)$ as the total amount of supply in the flow problem. Note the total amount of supply is preserved by any pre-flow routing, so $\sum_v f(v) = |\Delta(\cdot)|$ for any source-feasible pre-flow f.

3 Flow Algorithm

The main tool used in [12] is a local diffusion method that finds low conductance cuts, we use a flow based local method instead, which we describe in this section. Its basic building block is a unit flow method, which is used as a subroutine by an excess scaling flow algorithm. It produces either a pre-flow routing most of the source supply to sinks or a small conductance cut.

The unit flow method works on flow problems where $\forall v : \Delta(v) \leq wd(v)$ for constant $w \geq 2$. These flow problems are incremental in the sense that the initial excess supply on any v is not too large compared to its sink capacity $d(v)$, so intuitively it requires limited work to spread the excess supply to sinks. Additionally, since the primary concern is to find low conductance cuts, instead of routing as much supply to sinks as possible, we use a Push-Relabel algorithm [8], where we limit each label of a node to be at most $h = O(\ln m \ln \ln m)$ and we show that at termination either "enough" flow was routed or a low conductance cut with "large enough" volume can be found using a sweep cut method. These two aspects make the unit flow method very efficient.

We use excess scaling to divide a flow problem with a more general source supply function into multiple incremental phases that it solves using the unit flow method. The basic idea is as follows: We use a parameter μ, called *unit*, to scale down the source supply function such that a supply of x turns into x/μ units, each unit corresponding to a supply of μ. We choose the initial value μ large enough, so that after scaling down every $\Delta(v)$ by μ the source supply in unit μ satisfies $\forall v : \Delta(v) \leq 2d(v)$. Given the source supply function in unit μ, the unit flow method either returns a low conductance cut $(A, \bar{A})$, where $\min(\mathbf{vol}(A), \mathbf{vol}(\bar{A}))$ is "large", or it returns a flow that spreads out the supply so that a constant fraction of the total source supply is routed to vertices and each vertex v receives at most $d(v)$ units of supply. In the earlier case we terminate, in the later case we discard all source supply that we did not succeed in routing (and show that this only discards a constant fraction of the initial source supply in total) and then scale down μ by 2. Thus, in the new unit value, each vertex v has at most $2d(v)$ units of supply, which we use as source supply for the next unit flow invocation. Note that when we work in unit μ, the sink capacity of v is $d(v)$ units, i.e. $d(v)\mu$ supply in unit 1. Thus when μ is large, vertices have and transfer large amount of supply, which limits the volume of the subgraph that the unit flow procedure needs to explore to either send flow to or to find a low conductance cut in. As we decrease the value of μ geometrically, successive invocations of the unit flow method explore larger and larger subgraphs. This allows us to terminate early when there is a low conductance cut of small volume, and is the key to achieve local runtime.

3.1 Unit Flow The *Unit-Flow* subroutine (Algorithm 3.1) takes as input an undirected graph $G = (V, E)$ (with parallel edges but no self-loops), source function Δ and integer $w \geq 2$ such that $\forall v : 0 \leq \Delta(v) \leq wd(v)$, as well as an integer capacity $U > 0$ on all edges. Each vertex v is a sink of capacity $d(v)$. Furthermore, the procedure takes as input an integer $h \geq \ln(|E|)$ to customize the push-relabel algorithm, which we describe next.

In our push-relabel algorithm, each vertex v has a non-negative integer label $l(v)$ which is initially zero. The label of a vertex only increases during the execution of the algorithm and (in a modification of the standard push-relabel technique) cannot become larger than h. The bound of h on the labels makes the runtime of *Unit-Flow* linear in h, but it may prevent our algorithm from routing all units of supply to sinks even when there exists a feasible routing for the flow problem. However, when our algorithm cannot route a feasible flow, allowing labels of value up to h is sufficient to find a cut with low conductance (i.e., of value inversely proportional to h), which is our primary concern.

The algorithm maintains a pre-flow and the standard residual network, where each undirected edge $\{v, u\}$ in G corresponds to two directed arcs (v, u) and (u, v) in the residual network, with flow values such that $f(v, u) = -f(u, v)$, and $|f(v, u)|, |f(u, v)| \leq U$. The residual capacity of an arc (v, u) is $r_f(v, u) = U - f(v, u)$. We also maintain $f(v) = \Delta(v) + \sum_u f(u, v)$, which will be non-negative for all nodes v during the execution. The algorithm will explicitly enforce $f(v) \leq wd(v)$ for all v through the execution (i.e., it does not push flow to a vertex v if that would result in $f(v) > wd(v)$).

Algorithm 3.1.

```
Unit-Flow($G$,$\Delta$,$U$,$h$,$w$)
. Initialization:
. . $\forall\{v, u\} \in E$, $f(u, v) = f(v, u) = 0$.
. . $Q = \{v|\Delta(v) > d(v)\}$.
. . $\forall v$, $l(v) = 0$, and current($v$) is the first edge in
      its list of incident edges.
. While $Q$ is not empty
. . Let $v$ be the first vertex in $Q$, i.e. the lowest
      labelled active vertex.
. . Push/Relabel($v$).
. . If Push/Relabel($v$) pushes supply along $(v, u)$
. . . If $u$ becomes active, Add($u, Q$)
. . . If $v$ becomes in-active, Remove($v, Q$)
. . Else (i.e. Push/Relabel($v$) increases $l(v)$ by 1)
. . . If $l(v) < h$, Shift($v, Q$), Else Remove($v, Q$)
. . End If
. End While
```

```
Push/Relabel($v$)
. Let $\{v, u\}$ be current($v$).
. If Push($v, u$) is applicable, then Push($v, u$).
. Else
. . If $\{v, u\}$ is not the last edge in $v$'s list of edges.
. . . Set current($v$) to be the next edge in $v$'s list of
        edges.
. . Else (i.e. $(v, u)$ is the last edge of $v$)
. . . Relabel($v$), and set current($v$) to be the first
        edge of $v$'s list of edges.
. . End If
. End If
```

```
Push($v, u$)
. Applicability: $\mathrm{ex}(v) > 0, r_f(v, u) > 0$,
                 $l(v) = l(u) + 1$.
. Assertion: $f(u) < wd(u)$.
. $\psi = \min(\mathrm{ex}(v), r_f(v, u), wd(u) - f(u))$
. Send $\psi$ units of supply from $v$ to $u$:
    $f(v, u) \leftarrow f(v, u) + \psi, f(u, v) \leftarrow f(u, v) - \psi$.
```

```
Relabel($v$)
. Applicability: $v$ is active, and $\forall u \in V$,
                 $r_f(v, u) > 0 \implies l(v) \leq l(u)$.
. $l(v) \leftarrow l(v) + 1$.
```

As in the generic push-relabel framework, an *eligible* arc (v, u) is a pair such that $r_f(v, u) > 0$ and $l(v) = l(u) + 1$. A vertex v is *active* if $l(v) < h$ and $\mathrm{ex}(v) > 0$. The algorithm maintains the property that for any arc (v, u) with positive residual capacity, $l(v) \leq l(u) + 1$. The algorithm repeatedly picks an active vertex v with minimum label and either pushes along an eligible arc incident to v if there is one, or it raises the label of v by 1 if v is active if there is no eligible arc out of v.

Upon termination of the algorithm, we will have a pre-flow f, as well as labels $l(\cdot)$ on vertices. *Unit-Flow* will either successfully route a large amount of supply to sinks, or we can compute a low conductance cut using the labels. The proof is in Appendix B.

THEOREM 3.1. *Given* $G, \Delta, h, U, w \geq 2$ *such that* $\Delta(v) \leq wd(v)$ *for all* v*,* Unit-Flow *terminates with a pre-flow* f*, where we must have one of the following three cases*

(1) f *is a feasible flow, i.e.* $\forall v : \mathrm{ex}(v) = 0$*. All units of supply are absorbed by sinks.*

(2) f *is not a feasible flow, but* $\forall v : f(v) \geq d(v)$*, i.e., at least* $2m$ *units of supply are absorbed by sinks.*

(3) If f *satisfies neither of the two cases above, we can find a cut* $(A, \bar{A})$ *such that* $wd(v) \geq f(v) \geq d(v)$ *for all* $v \in A$*, and* $f(v) \leq d(v)$ *for all* $v \in \bar{A}$*. Furthermore*

  (a) If $h \geq \ln m$*, the conductance* $\Phi(A) = \frac{|E(A,V\setminus A)|}{\min(\mathbf{vol}(A),2m-\mathbf{vol}(A))} \leq \frac{20 \ln 2m}{h} + \frac{w}{U}$*.*

  (b) If $h = \Omega(\ln m' \ln\ln m')$ *for* $m' \geq m$*, we have a more fine-grained conductance guarantee: let* K *be the smaller side of* $(A, \bar{A})$*, then* $\Phi(K) \leq \frac{\ln m+1-\lceil \ln \mathbf{vol}(K)\rceil}{50 \ln m'} + \frac{w}{U}$*.*

The motivation for maintaining $f(v) \leq wd(v)$ throughout the algorithm is to establish lower bounds on $\mathbf{vol}(A)$. Intuitively, if the total amount of excess supply is large at the end, $\mathbf{vol}(A)$ must be large as no single vertex can have too much excess. More specifically, we have the following observations.

OBSERVATION 3.1. *If the output fulfills case* (3) *of Theorem 3.1, we have* $\sum_{v\in V} \mathrm{ex}(v) \leq (w - 1)\mathbf{vol}(A)$

OBSERVATION 3.2. *When* $|\Delta(\cdot)| \geq tm$ *for constant* $t > 2$*, we must have* $\sum_{v\in V} \mathrm{ex}(v) \geq (t - 2)m$*. If* w *is a constant, and we get case* (3)*, then every node* v *in* A *absorbs* $d(v)$ *units of supply, and* $\mathbf{vol}(A) = \Theta(m)$*.*

Unit-Flow returns a pre-flow f and a possibly empty cut A. Additionally, we treat units of supply as distinct tokens with marks bearing information, which must be preserved by the pre-flow, leading to an extra $O(1)$ work per push of a single unit. This leads to the following running time result, whose proof is in Appendix B.

LEMMA 3.1. *The running time for* Unit-Flow *is* $O(w|\Delta(\cdot)|h)$*.*

3.2 Excess Scaling Flow Algorithm The excess scaling procedure (Algorithm 3.2) takes as input an undirected graph G (with parallel edges) of volume $2m$, source function Δ such that $|\Delta(\cdot)| = 2m$, constant $\tau \in (0, 1)$, capacity parameter U, and an integer $h \geq \ln m$. Recall that each vertex v is a sink of capacity $d(v)$. The algorithm will either in time $O(mh)$ route at least $(1-\tau)2m$ supply to sinks, or find a low conductance cut $(A, \bar{A})$ in time proportional to $\min(\mathbf{vol}(A), \mathbf{vol}(\bar{A}))$.

The procedure divides the flow problem into incremental phases, and uses successive *Unit-Flow* invocations on them. This is done via a parameter μ, which is the value of one unit in *Unit-Flow*. Initially, $\mu = \max_v \frac{\Delta(v)}{2d(v)}$ such that each v has initial source supply at most $2d(v)$ units. It then calls *Unit-Flow* with scaled source function Δ/μ and $w = 2$. Every unit of supply in *Unit-Flow* is supply of value μ in the original problem. To avoid confusion, when we say x *supply*, we mean a supply of value x, and when we say x *units of supply*, we mean a supply of value $x\mu$. Algorithm 3.2 calls *Unit-Flow* repeatedly with a geometrically decreasing value of μ. The sink capacity of v is $d(v)$ units in *Unit-Flow*, but the pre-flow returned by *Unit-Flow* may have excess supply on vertices. To use the supply on vertices at the end of a *Unit-Flow* invocation as the source supply of the next *Unit-Flow* call, we simply discard all excess supply (as we show this will only discard a small fraction of the total supply). Then there is at most $d(v)$ supply in unit μ at each vertex v. Thus we can halve the value of μ so that each v has at most $2d(v)$ supply in the new unit. If, however, every node v has at most $d(v)$ supply in unit 1, we terminate as each vertex can absorb its supply.

From a flow point of view in the j-th call to *Unit-Flow* for $j = 0, 1, ...$ each node v has a source supply $\Delta_j(v)$, where $\Delta_0(v) = \Delta(v)$ and for $j > 0$, $\Delta_j(v) = \mu \cdot \min(d(v), f_{j-1}(v))$ (the min captures the removal of excess supply), where $f_{j-1}(v)$ is the amount of supply ending at v after the $j-1$-st call to *Unit-Flow*. Assume for the moment that $f_{j-1}(v) \leq d(v)$. Then for $j > 0$, $\Delta_j(v) = \mu \cdot f_{j-1}(v)$, i.e., each node v has as source supply in the j-th call to *Unit-Flow* exactly the supply values that it received in the previous call. Thus, no supply is absorbed at nodes between consecutive calls of *Unit-Flow*, the supply is just “spread out” more and more. Once the supply ending at each node is at most its degree, the procedure terminates. Due to the removal of excess supply this happens for sure when $\mu = 1$, but it might already happen for a larger value of μ. As the final flow f is the sum of all flows f_j and each call to *Unit-Flow* uses at most $U\mu$ edge capacity with μ geometrically decreasing, each edge carries at most $2UF$ flow. As the total source supply given to the j-th call is $|\Delta_j(\cdot)/\mu| \leq 2m/\mu$, its runtime is $O(mh/\mu)$ and as μ decreases geometrically the total time for all calls to *Unit-Flow* is $O(mh/\mu_f)$, where μ_f is the value of μ at termination.

Algorithm 3.2 returns a pre-flow f, a possibly empty cut A, and a function $\Delta'(\cdot)$ on vertices such that $\Delta'(v)$ is the amount of the $\Delta(v)$ source supply starting at v that is routed to sinks at the end, i.e. never removed as excess supply. Since we can mark the supply with the original source vertex, and the invocations of *Unit-Flow* maintain the marks, $\Delta'(\cdot)$ will be easy to compute.

Algorithm 3.2. Excess scaling flow procedure

```
. **Input**: $G = (V, E)$, $\Delta(\cdot)$, $\tau$, $U$, $h$.
. **Initialization**: Let $F = \max_v \frac{\Delta(v)}{2d(v)}$, $\mu = F$, $j = 0$,
   $\Delta_0 = \Delta' = \Delta$, $f$ be zero pre-flow
. **Repeat**
. . *Note:* $\Delta_j(v) \leq 2d(v)\mu$
. . Run *Unit-Flow*$(G, \frac{\Delta_j(v)}{\mu}, U, h, w = 2)$.
     Get back $f_j$ in unit $\mu$, and $A_j$.
. . *Add $f_j$ to our current preflow:*
     $\forall(v, u) \quad f(v, u) \leftarrow f(v, u) + f_j(v, u)\mu$.
. . *Remove excess supply on vertices:*
     $\forall v \quad \Delta_{j+1}(v) = (f_j(v) - \mathrm{ex}_j(v))\mu$. Update $\Delta'$.
. . **If vol**$(A_j) \geq \frac{\tau 2m}{10\mu \ln 2\mu \ln\ln m}$: **Return** $f$, $\Delta'$, and
     $A \overset{\mathsf{def}}{=}$ smaller side of $A_j, \bar{A}_j$. **Terminate.**
. . **If** $\max_v \Delta_{j+1}(v)/d(v) \leq 1$: **Return** $f$, $\Delta'$, and
     $A \overset{\mathsf{def}}{=} \emptyset$. **Terminate.** // Supply at most $d(v)\forall v$
. . $\mu \leftarrow \mu/2$, $j \leftarrow j + 1$, proceed to next iteration.
. **End Repeat**
```

Each *Unit-Flow* invocation returns a possibly empty low conductance cut. If at any point the volume of the returned cut is large compared to the total work done so far, the algorithm can terminate with a low conductance cut $(A, \bar{A})$ in time $\tilde{O}(h\mathbf{vol}(A))$, i.e., in " local" time. If this never happens, since the volume of the cut returned after each *Unit-Flow* upper-bounds the amount of removed excess supply (Observation 3.1), the algorithm must route at least $(1 - \tau)2m$ supply to sinks at the end. Formally, we have the following result, whose proof is in Appendix B.

Lemma 3.2. *Given a graph G of volume $2m$, a source function Δ such that $|\Delta(\cdot)| = 2m$, a constant $0 < \tau < 1$, and positive parameters U and h, the flow procedure will return a preflow f, subject to edge capacity of $2UF$ on every edge, where $F = \max_v \frac{\Delta(v)}{2d(v)}$. It will also return $\Delta'(\cdot)$, the amount of source supply from each vertex that is routed to sinks, where each v is a sink of capacity $d(v)$. In addition, we have either of the two cases below:*

(1) At least a $(1 - \tau)$ fraction of the total source supply is routed to sinks

$$|\Delta'(\cdot)| \geq (1 - \tau)2m$$

The runtime is $O(mh)$ in this case.

(2) It returns a cut $(A, \bar{A})$, with $\mathbf{vol}(A) \leq \mathbf{vol}(\bar{A})$, and $\mathbf{vol}(A)$ is $\Omega(\frac{m}{F \ln m \ln\ln m})$. The runtime is $O(\mathbf{vol}(A) h \ln \frac{m}{\mathbf{vol}(A)} \ln\ln m)$. Furthermore

(a) If $h \geq \ln m$, $\Phi(A) \leq \frac{20 \ln 2m}{h} + \frac{2}{U}$.

(b) If $h = \Omega(\ln m' \ln\ln m')$ with $m' \geq m$, $\Phi(A) \leq \frac{(\log m + 1 - \lceil \log \mathbf{vol}(A) \rceil)}{20 \log m'} + \frac{2}{U}$

4 The Kawarabayashi-Thorup decomposition framework

In the rest of the paper, we show how to modify the algorithm in [12] (the K-T algorithm) to use the efficient flow procedures in Section 3, and eventually get a $O(m \ln^2 m \ln\ln^2 m)$ algorithm. We divide the K-T algorithm (Algorithm A.1 in Appendix A) into two layers: the inner procedure which we replace with our own, and the K-T framework (i.e. everything outside the inner procedure) which we do not change. We have a clean interface between the two, which is formally presented as Theorem 5.1. We will discuss the K-T framework and the interface in this section. Since this section is all about material in [12], we keep our discussion at a very high level. For completeness, we include the details in Appendix A

Given an undirected simple graph $G = (V, E)$ with minimum degree δ, the decomposition framework computes a (multi-)graph with $O(\frac{m_G \ln m_G}{\delta})$ edges, while preserving all non-trivial min cuts of G. Note that δ upperbounds the value of the min cut, and when δ is $O(\ln m_G)$, G has $O(\frac{m_G \ln m_G}{\delta})$ edges already, so we only work on the case of $\delta = \Omega(\ln m_G)$.

The high-level approach is to start with $\overline{G} = G$, and recursively contract subsets of nodes into *supervertices* to reduce the size of $\overline{G}$ (the outer loop in Algorithm A.1). Throughout the algorithm, a node (or vertex) in $\overline{G}$ is either a regular vertex (i.e. a vertex in G) or a supervertex (i.e. a subset of vertices of V). At any point, the supervertices (as subsets of V) and the regular vertices (as singleton sets) in $\overline{G}$ give a partition of V. All edges of G, except those collapsed into supervertices, are in $\overline{G}$. In particular, any regular vertex in $\overline{G}$ has degree at least δ.

In each iteration of the outer loop, the algorithm computes (disjoint) subsets of nodes in $\overline{G}$ that can be contracted. More specifically, we maintain H, with $H = \overline{G}$ at the start of the iteration, edges and nodes will be removed through the iteration, and at the end, H will be a collection of connected components such that each component must fall entirely on one side of any minimum cut, and, thus, can be contracted.

At the start of the iteration, supervertices with degree less than $c_1\gamma\delta$ (called *passive* supervertices) are removed from H, where c_1 is a suitably chosen constant, and $\gamma = \Theta(\ln m)$. Throughout the iteration, whenever the algorithm removes edges and nodes from H, it will also *trim* H, which is to recursively remove from H any node that has lost more than $\frac{3}{5}$ of its degree (comparing to its degree in $\overline{G}$). In particular, every connected component C in H will be *trimmed*, i.e. $\forall v \in C : d_C(v) \geq \frac{2}{5}d_{\overline{G}}(v)$. To find components that can be contracted, we first find *clusters*.

Definition 4.1. *A trimmed subset C is a* cluster *if for every cut of cut-size at most δ in $\overline{G}$, one side contains at most 2 regular vertices and no supervertex from C.*

A cluster component is a component that almost entirely falls in one side of any minimum cut. Given a cluster component it is easy to get its *core*, which is the part that can be contracted (See Appendix A). Thus the major work is to find cluster components. We have the following measure of how close C is to a cluster.

Definition 4.2. *A connected component C of H is s-*strong *if every cut $(S, \bar{S})$ of $\overline{G}$ with cut-size at most δ satisfy $\min(\mathbf{vol}_C(S \cap C), \mathbf{vol}_C(\bar{S} \cap C)) \leq s$. We call s the strength of C.*

Informally, the smaller the strength of C, the closer C is to fall entirely in one side of any minimum cut. Note a component C is by definition m_C-strong, and any subcomponent of a s-strong component is also s-strong. The strategy of the algorithm is to drive down the strengh of the components in H, and the following is a sufficient condition to have a cluster.

Lemma 4.1. *Let $s_0 = 1000\gamma\delta$, any trimmed s_0-strong connected component C in H is a cluster.*

To get components of smaller strength, the algorithm relies on the inner procedure (See Theorem 5.1). Each time the inner procedure is invoked, it is given a trimmed component C in H that is already certified to be s-strong for some $s > s_0$. The inner procedure will either certify that C is $0.6s$-strong, or find a low conductance cut in C. In the latter case, we can remove the cut edges from H, and break C into smaller components. This is useful since the volume of a component is a trivial bound on its strength. The low conductance is crucial, as we need to bound the total number of cut edges removed during the entire process. Ultimately, a constant fraction of the edges from $\overline{G}$ will remain in the connected components in H, which are contracted at the end, so the volume of $\overline{G}$ drops geometrically across outer loop iterations (See Lemma A.5 and Lemma A.4 for details).

The runtime of each invocation of the inner procedure is proportional to the progress made in that invocation. That is, if it only finds a low conductance cut $(A, \bar{A})$, the runtime is local. If, however, the inner procedure spends $\tilde{O}(m_C)$ runtime on a component C, it certifies a smaller strength for C (or a subcomponent of volume $\Theta(\mathbf{vol}(C))$ of C). See Section 6 for more details.

5 The inner procedure

In this section we give a high level descriptions of the inner procedure (See Appendix C for details). We follow a similar approach as [12], but use the flow methods in Section 3 instead of diffusions as subroutines. As discussed above, the K-T framework relies on the inner procedure to achieve the following.

Theorem 5.1. *Given an s-strong trimmed component C with $m_C \leq s \leq s_0 = 1000\gamma\delta$, the inner procedure will achieve one of the following:*

1. *Find a set A with $\mathbf{vol}(A) \leq m_C$, and*
$$\Phi(A) \leq \frac{(\log m_C + 1 - \lceil \log \mathbf{vol}(A) \rceil)}{20 \log m_G}$$
in time $O(\mathbf{vol}(A) \ln \frac{m_C}{\mathbf{vol}(A)} \ln m_G \ln\ln^2 m_G)$.
2. *Find a set A with*
$$\Phi(A) \leq \frac{(\log m_C + 1 - \lceil \min(\log \mathbf{vol}(A), \log \mathbf{vol}(\bar{A})) \rceil)}{20 \log m_G}$$
in time $O(m_C \ln m_G \ln\ln m_G)$. Moreover, $\mathbf{vol}(A)$ is $\Theta(m_C)$, and A is certified to be $0.6s$-strong.
3. *Certify that C is $0.6s$-strong in time $O(m_C \ln m_G \ln\ln m_G)$.*

The intuition is as follows. If C is a connected component of H that is not a cluster, there must exist cuts in C of cut-size at most δ. Consider any such small cut and denote by S the side with minimum volume. We know $\mathbf{vol}_C(S) \leq s$, since C is s-strong. The cut-size being at most δ gives a strong bottleneck to route into or out of S, and we can exploit this bottleneck.

Use the cut as a bottleneck to route supply out of S. The excess-scaling algorithm (Algorithm 3.2) guarantees to find a low conductance cut in local runtime if we give it a very infeasible flow problem, i.e. one where it is impossible to route a large fraction of the source supply to sinks (Lemma 3.2). A small cut S naturally gives such very infeasible flow problems as follows. As the total sink capacity in S is $\mathbf{vol}_C(S)$, the condition $\mathbf{vol}_C(S) \leq s$ bounds the total sink capacity in S by at most s. As there are at most δ edges on the cut, we can pick an appropriate edge capacity parameter to get a good bound on the cut capacity of S. As long as we choose a source function such that the source supply in S is large, for example twice the sum of S's sink and cut capacity, we get a very infeasible flow problem. The difficulty, however, is to construct such a source function *without knowing* S. The strategy, very informally, is as follows. We construct a large number ($\leq$ 5000) of flow problems with different source functions, and run (in parallel, step by step) Algorithm 3.2 on them, terminating them whenever one of them returns a low conductance cut, or, if this does not happen, letting them all run to termination. If any of these flow problems had large enough source supply in S, we get a low conductance cut in local runtime, i.e., case (1) in Theorem 5.1.

Use the cut as a bottleneck to route supply into S. If we do not get the above case, we end with case (1) of Lemma 3.2 for all the flow problems we constructed, and we have spent $O(m_C \ln m)$ time for them. In this case, we know that the source functions of these flow problems all have little source supply starting in S as they were able to route most of their flow to a sink. Using the $\Delta(v)'$ values returned by each execution of Algorithm 3.2, we suitably combine the successfully routed source supplies to a new, well spread-out source function. More specifically, the new source function fulfills the following properties: (a) Very little source supply is in S, and the cut bounds the amount of supply that can be pushed into S, so the total supply ending in S must be small. (b) The amount of total supply is large (more formally at least $4m_C$) and well spread out (more formally $\forall v : \Delta(v) \leq wd(v)$). Thus we can run a *Unit-Flow* computation directly on it (*without* going through the excess scaling procedure). We use $h = \Theta(\ln m_G \ln\ln m_G)$ and $w = 25$ and have either of the two outcomes below.

(A) All nodes in C have their sinks saturated (case (2) of Theorem 3.1). Since the amount of supply ending in S is small, the total sink capacity in S must also be small, i.e. $\mathbf{vol}_C(S)$ must be small. Recall S is any cut in C with cut-size at most δ. Thus we know all such cuts have small volume, more specifically at most $0.6s$, implying that C is $0.6s$-strong, i.e. case (3) of Theorem 5.1.

(B) We get a set A as specified in case ($3b$) of Theorem 3.1. Since all nodes in A have their sinks saturated, by a similar argument as above, we show that $\mathbf{vol}_C(A \cap S)$ is small. Again as this argument works for any S of cut-size at most δ, we can argue A is $0.6s$-strong. Additionally, case (3b) of Theorem 3.1 and Observation 3.2 give the desired bound on $\Phi(A)$ and show that $\mathbf{vol}(A) = \Theta(m_C)$, which shows that case (2) of Theorem 5.1 holds.

Note that the outline of the flow problem construction is similar to the seeding of diffusions in [12], but the details differ in part due to their ability to use the linearity property of diffusions. We must explicitly spread out our flows and warm start our procedures in some cases as noted above.

6 Running time analysis

To compute the edge connectivity of an undirected simple graph G with m_G edges, we first construct $\overline{G}$ as discussed earlier, and use Gabow's min-cut algorithm [7] on $\overline{G}$. We start with the runtime to construct $\overline{G}$. Recall we use the K-T framework (Algorithm A.1) with our flow based inner procedure (Algorithm C.1). By Lemma A.5, $m_{\overline{G}}$ decreases geometrically across iterations of the outer loop. As the runtime of each outer loop iteration will be $\Omega(m_{\overline{G}})$, the first iteration will dominate asymptotically, so we focus on the first iteration, with $m_{\overline{G}} = m_G$.

The operations outside of the middle loop in total take $O(m_{\overline{G}})$ time (see details in Appendix A). To analyze the middle loop, we look at each invocation of the inner loop. Informally we will charge the runtime to edges such that an edge is charged when it lies in the smaller side of a cut, or the strength of its component drops by a constant factor. More specifically, given an s-strong component C in H, we have three cases by Theorem 5.1.

(1) Find a cut $(A, C \setminus A)$ with $\mathbf{vol}(A) \leq m_C$ in time $O(\mathbf{vol}(A) \ln(m_C/\mathbf{vol}(A)) \ln m_G \ln\ln^2 m_G)$. We can charge $O(\ln(\mathbf{vol}(C)/\mathbf{vol}(A)) \ln m_G \ln\ln^2 m_G)$ to each edge in A.

Consider the total charge to any edge by all invocations of inner procedure of this case. The edge is charged when it falls in the smaller side of a cut. The $\ln \frac{\mathbf{vol}(C)}{\mathbf{vol}(A)}$ part will telescope, so in total each edge is charged $O(\ln^2 m_G \ln\ln^2 m_G)$.

(2) Find a subset A in C where $\mathbf{vol}(A)$ is $\Theta(m_C)$, and A is certified to be $0.6s$-strong. The runtime is $O(m_C \ln m_G \ln\ln m_G)$. We can charge $O(\ln m_G \ln\ln m_G)$ to each edge in A.

Over all invocations of inner procedure of this case, any edge is charged at most $O(\ln m_{\overline{G}})$ times, since the strength of its component decreases geometrically each time we charge the edge. In total each edge is charged $O(\ln^2 m_G \ln\ln m_G)$.

(3) Certify the entire component C is $0.6s$-strong. The runtime is $O(m_C \ln m_G \ln\ln m_G)$. We use the same argument as in case (2) above.

In total, we can charge the runtime of the middle loop to the edges in $\overline{G}$, and each edge is charged $O(\ln^2 m_G \ln\ln^2 m_G)$, so the runtime is $O(m_G \ln^2 m_G \ln\ln^2 m_G)$.

At the end, we get a multi-graph $\overline{G}$ with $O(\frac{m_G \ln m_G}{\delta})$ edges, preserving all non-trivial min cuts of G. We use Gabow's min-cut algorithm [7] on $\overline{G}$. Gabow's algorithm works on multi-graphs, and takes time $O(\lambda m_{\overline{G}} \ln m_{\overline{G}})$ on $\overline{G}$, where λ is the size of the min cut. With our bound on $m_{\overline{G}}$, as well as $\lambda \leq \delta$, the runtime of Gabow's algorithm is thus $O(m_G \ln^2 m_G)$. Together with the runtime to construct $\overline{G}$, we have the following.

Theorem 6.1. *The minimum cut in an undirected simple graph with m edges can be computed in time $O(m \ln^2 m \ln\ln^2 m)$.*

References

[1] R. Andersen, F. R. K. Chung, and K. J. Lang, *Local graph partitioning using pagerank vectors*, in 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings, IEEE Computer Society, 2006, pp. 475–486.

[2] R. Andersen and Y. Peres, *Finding sparse cuts locally using evolving sets*, in Proceedings of the forty-first annual ACM symposium on Theory of computing, ACM, 2009, pp. 235–244.

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

[4] F. R. K. Chung, *A local graph partitioning algorithm using heat kernel pagerank*, Internet Mathematics, 6 (2009), pp. 315–330.

[5] M. B. Cohen, R. Kyng, G. L. Miller, J. W. Pachocki, R. Peng, A. Rao, and S. C. Xu, *Solving SDD linear systems in nearly* mlog$^{1/2}$n *time*, in Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, 2014, pp. 343–352.

[6] A. Frank, *On the edge-connectivity algorithm of nagamochi and ibaraki*, Laboratoire Artemis, IMAG, Université J. Fourier, Grenoble, (1994).

[7] H. N. Gabow, *A matroid approach to finding edge connectivity and packing arborescences*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA, C. Koutsougeras and J. S. Vitter, eds., ACM, 1991, pp. 112–122.

[8] A. V. Goldberg and R. E. Tarjan, *Efficient maximum flow algorithms*, Commun. ACM, 57 (2014), pp. 82–89.

[9] R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, Journal of the Society for Industrial and Applied Mathematics, 9 (1961), pp. 551–570.

[10] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh, *Wtf: The who to follow service at twitter*, in Proceedings of the 22Nd International Conference on World Wide Web, WWW '13, New York, NY, USA, 2013, ACM, pp. 505–514.

[11] D. R. Karger, *Minimum cuts in near-linear time*, J. ACM, 47 (2000), pp. 46–76.

[12] K. Kawarabayashi and M. Thorup, *Deterministic global minimum cut of a simple graph in near-linear time*, in Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015, R. A. Servedio and R. Rubinfeld, eds., ACM, 2015, pp. 665–674.

[13] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, *A simple, combinatorial algorithm for solving SDD systems in nearly-linear time*, in Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013, 2013, pp. 911–920.

[14] R. Khandekar, S. Rao, and U. V. Vazirani, *Graph partitioning using single commodity flows*, J. ACM, 56 (2009).

[15] L. Lovász and M. Simonovits, *Random walks in a convex body and an improved volume algorithm*, Random Struct. Algorithms, 4 (1993), pp. 359–412.

[16] D. W. Matula, *A linear time 2+epsilon approximation algorithm for edge connectivity*, in Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas., V. Ramachandran, ed., ACM/SIAM, 1993, pp. 500–504.

[17] H. Nagamochi and T. Ibaraki, *Computing edge-connectivity in multiple and capacitated graphs*, in Algorithms, International Symposium SIGAL '90, Tokyo, Japan, August 16-18, 1990, Proceedings, T. Asano, T. Ibaraki, H. Imai, and T. Nishizeki, eds., vol. 450 of Lecture Notes in Computer Science, Springer, 1990, pp. 12–20.

[18] L. Orecchia, L. J. Schulman, U. V. Vazirani, and N. K. Vishnoi, *On partitioning graphs via single commodity flows*, in Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008, C. Dwork, ed., ACM, 2008, pp. 461–470.

[19] L. Orecchia and Z. A. Zhu, *Flow-based algorithms for local graph clustering*, in Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on

Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, C. Chekuri, ed., SIAM, 2014, pp. 1267–1286.

[20] D. A. Spielman and S. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004, L. Babai, ed., ACM, 2004, pp. 81–90.

[21] M. Stoer and F. Wagner, *A simple min-cut algorithm*, Journal of the ACM (JACM), 44 (1997), pp. 585–591.

[22] Z. A. Zhu, S. Lattanzi, and V. S. Mirrokni, *A local algorithm for finding well-connected clusters*, in Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013, vol. 28 of JMLR Proceedings, JMLR.org, 2013, pp. 396–404.

A Details and analysis of Section 4

A.1 K-T framework In this section we discuss the K-T framework (Algorithm A.1) with more details. Since the definitions and lemmata in this part are restated from [12], sometimes with slightly modified parameters, we omit the proofs in our presentation, and refer interested readers to [12].

Given an undirected simple graph G with minimum degree δ, the decomposition framework produces a graph, $\overline{G}$, with $O(\frac{m_G \ln m_G}{\delta})$ edges, where any non-trivial minimum cut in G corresponds to a minimum cut in $\overline{G}$. Note that when δ is $O(\ln m_G)$, the algorithm will call Gabow's algorithm [7] directly without sparsifying G.

The high-level approach is to start with $\overline{G} = G$, and recursively contract subsets of nodes in $\overline{G}$ into *supervertices* to reduce the size of the graph, while preserving all non-trivial minimum cuts of G. Throughout the algorithm we maintain the multi-graph $\overline{G} = (\overline{V}, \overline{E})$, where a node (or interchangeably, a vertex) in $\overline{G}$ is either a regular, non-contracted vertex or a supervertex. We consider a supervertex to be both a node of $\overline{G}$ and a set of vertices of V. The set of vertices contained in different supervertices of $\overline{G}$ are disjoint, and supervertices can be further contracted with other vertices during the course of the algorithm. With the formation of supervertices, we have parallel edges in $\overline{G}$. Since non-trivial minimum cuts in G have cut-size at most δ, the algorithm periodically contracts vertices with more than δ parallel edges between them.

The multi-graph $\overline{G}$ has all the edges of G, except those whose both endpoints belong to the same supervertex. Thus a regular vertex in $\overline{G}$ will have the same number of incident edges as in G, so its degree in $\overline{G}$ is at least δ. The degree of a supervertex is the number of edges incident to it in $\overline{G}$, or equivalently the number of edges in G incident to exactly one node in the set of regular vertices contracted in the supervertex.

In each iteration of the outer loop, the algorithm computes (disjoint) subsets of nodes in $\overline{G}$ that can be contracted. More specifically, we maintain H, with $H = \overline{G}$ at the start of the iteration, edges and nodes will be removed through the iteration, and at the end, H will be a collection of connected components such that each component must fall entirely on one side of any minimum cut, and, thus, can be contracted.

At the start of the iteration, supervertices with degree less than $c_1\gamma\delta$ (called *passive* supervertices) are removed from H, where c_1 is a suitably chosen constant, and $\gamma = \Theta(\ln m)$. Throughout the iteration, whenever the algorithm removes edges and nodes from H, it will also *trim* H. The *trimming* operation of a subgraph H of $\overline{G}$ is to recursively remove from H any (regular or super-) vertex v that has $d_H(v) \leq \frac{2}{5}d_{\overline{G}}(v)$. Said differently, when we *trim* a subgraph H, we recursively remove every vertex v in H that has "lost" (in comparison to $\overline{G}$) at least $\frac{3}{5}d(v)$ of its edges Furthermore, a subset C of nodes in $\overline{G}$ is *trimmed* if $d_C(v) \geq \frac{2}{5}d_{\overline{G}}(v)$ for all $v \in C$ (where $d_C(v)$ is counting edges in $\overline{G}$, not in H), i.e. at least a $\frac{2}{5}$-th fraction of v's incident edges in $\overline{G}$ are internal to C. Note that after trimming a subgraph H, every connected components C in H is trimmed as for the fact that C is a connected component in H it follows that each vertex in $v \in C$ must have $d_C(v) = d_H(v)$.

Our goal is to detect subsets C of nodes that can be contracted to supervertices. Thus for each such subset C there cannot be a non-trivial minimum cut S of $\overline{G}$ such that $|S \cap C| > 1$ and $|\bar{S} \cap C| > 1$. We call this condition on a set of nodes ***Requirement R1***. To find components that can be contracted, the algorithm first finds *clusters* (See Definition 4.1). A cluster is a component that almost entirely falls in one side of any minimum cut. Once we have a cluster, it will be easy to get a *core* of the cluster, that can be contracted.

Construction of the core for a cluster: We now distill from each cluster connected component C a subset of nodes that fullfils R1. To do so we take every vertex v in C that has degree at least $\frac{1}{2}d_{\overline{G}}(v) + 1 \geq \delta/2+1$ in H. More formally, we say a node v in a cluster C is *loose* if it is a regular vertex, and $d_C(v) \leq d_{\overline{G}}(v)/2$, that is, at most $d_{\overline{G}}(v)/2$ of its edges in $\overline{G}$ go to neighbors in C. By *shaving* we refer to the operation of removing loose vertices of a cluster from H. Note that shaving only depends on the degree of the nodes in $\overline{G}$, not in H. The problem created by shaving is, however, that the remaining nodes of C might no longer be connected, in which case C does not fulfill R1, but every connected component A that is created by shaving C will actually fulfill R1, thus can be contracted. Although all such shaved cluster components contain no minimum cut, the algorithm will only contract A when it has a large volume, as otherwise we will end up with too many supervertices at the end. Formally, we introduce the notation $\mathbf{ivol}(A, C)$ with $A \subseteq C$ as the number of edges with one endpoint in A and one endpoint in $C \setminus A$ *plus* twice the number of edges with both endpoints in A. A shaved cluster connected component A will be a *core* of C if $\mathbf{ivol}_{\overline{G}}(A, C) \geq \mathbf{vol}_{\overline{G}}(C)/4$. By *scraping* we mean the operation of removing the entire connected component A from H if A is not a core of the cluster C. The algorithm will contract all the nodes in A when A is a core of a cluster C.

Algorithm A.1. Kawarabayashi-Thorup framework

```
$\overline{G} \leftarrow G$; $G$ has min degree $\delta \geq \gamma = \Theta(lnm_G)$
Repeat (Outer loop)
. $H \leftarrow \overline{G}$.
. Remove passive supervertices from H and trim H.
. While any connected component $C$ in $H$ is not a
    certified cluster do (Middle loop)
. . Let $s$ be the smallest integer such that $C$ is
      certified $s$-strong.
. . Inner procedure:
      Achieve either (1), (2), or (3) (Theorem 5.1)
. . . (1) Find low conductance cut $(A, \bar{A})$ of $C$ in
          time $\tilde{O}(\min(\mathbf{vol}(A), \mathbf{vol}(\bar{A}))$.
. . . (2) Find low conductance cut $(A, \bar{A})$ of $C$ in
          time $\tilde{O}(\mathbf{vol}(C))$ where $A$ is $0.6s$-strong in $H$
          and $vol(A)$ is $\Theta(\mathbf{vol}(C))$ .
. . . (3) Certify that $C$ is $0.6s$-strong in $H$ in time
          $\tilde{O}(\mathbf{vol}(C))$.
. . End inner procedure
. . If a cut was found (case (1)(2)), remove the cut
      edges from $H$ and trim $H$.
. End while
. Take each cluster component of $H$, and contract its
    core to a supervertex in $\overline{G}$.
. Contract any two vertices that have more than $\delta$
    parallel edges between them.
Until $\geq \frac{1}{20}$ of the edges in $\overline{G}$ are incident to passive
supervertices.
```

To find the clusters, we use the *strength* (See Definition 4.2) as the measure of how close a component C is to a cluster. The strategy of the algorithm is to drive down the strengh of the components in H.

Recall we maintain H that is a subgraph of $\overline{G}$. All passive supervertices are removed from H at the beginning, and H is trimmed. As a consequence of these specific operations on H, the following is a sufficient condition to have a cluster in H.

Lemma 4.1. *Let $s_0 = 1000\gamma\delta$, any trimmed s_0-strong connected component C in H is a cluster.*

The structure of the algorithm is as follows: It consists of three nested loops, which we call (a) the outer loop, (b) the middle loop, and (c) the inner procedure. Both, the algorithm of [12] and our algorithm use this structure, however, we differ in the implementation of the inner procedure.

A.2 Analysis

In this section we show the correctness of the K-T framework (Algorithm A.1). To prove the correctness we show (1) the termination of the middle loop, (2) the termination of the outer loop, (3) that no mincut of size at most δ is contained in a core, and (4) that the resulting graph $\overline{G}$ contains $O(m_G \ln m_G/\delta)$ edges.

To guarantee the termination of the middle loop we have to show that at some point all connected components are clusters or individual nodes. Started on an s-strong component C in H, each iteration of the middle loop either reduces the size of C (and potentially shows that one of the new connected components is $0.6s$-strong in H) or shows that C is $0.6s$-strong in H, i.e., either reducing the size of C or its strength. Note that removing edges of H does not increase the strength of its components, i.e., any component that was s-strong in the old H is also s-strong in the new H. Together with Lemma 4.1, eventually every connected component in H must either has size 1 or is a cluster, so the middle loop will always terminate.

The proof of Lemma 4.1 crucially relies on the removal of passive supervertices and the trimming of H. The high level intuition is as follows. Consider component C in H, and cut $(S, \bar{S})$ of cut-size at most δ. Let U be the smaller side of $S \cap C$ and $\bar{S} \cap C$ (in terms of $\mathbf{vol}_C$). If U contains any supervertex, then $\mathbf{vol}_C(U) > s_0$, since every active supervertex has large degree. If U contains only regular vertices, each regular vertex has degree at least $\frac{2}{5}\delta$, as C is trimmed. When U has at least 3 nodes, the total degree is at least $\frac{6}{5}\delta$, and at most δ of these degree can go from U to $C \setminus U$. Thus U must has $\Omega(\delta)$ nodes to accomodate these degree, as there is no parallel edges between regular vertices. This again gives $\mathbf{vol}_C(U) > s_0$.

The while loop (middle loop) terminates when all connected components remaining in H are clusters, and as discussed earlier, the algorithm contracts the cores of the clusters into supervertices. The following lemma shows every core has the desired property of containing no non-trivial cut of size at most δ (see also Lemma 14 in [12]).

Lemma A.1. *No core contracted by the algorithm contains a non-trivial mincut cut of G.*

The intuition is as follows. As C is a cluster we know that for every cut S of cut-size at most δ in $\overline{G}$, one side contains at most 2 regular vertices and no supervertex from C. Assume a cut S where there are exactly 2 regular vertices on one side of the cut. As each vertex has at least $\delta/2+1$ edges going to nodes in C, it follows that at least $2(\delta/2+1)-1 > \delta$ edges are crossing the cut S, i.e., S cannot be a minimum cut. We will choose c_1 such that a similar argument shows that S is not a non-trivial minimum cut if supervertices belong to both sides of the cut S. Thus it follows that for each non-trivial minimum cut either $A \cap S = \emptyset$ or $A \cap \bar{S} = \emptyset$, i.e., A contains no non-trivial minimum cut.

The above argument shows no shaved cluster A contains any non-trivial minimum cut. However, the algorithm only contracts A when it is a core, since otherwise we may end up with too many supervertices. Recall A is a core when it has large internal volume, and formally it can be shown that every supervertex has $\Omega(\delta^2)$ volume contracted inside it. This will bound the total number of supervertices by $O(\frac{m}{\delta^2})$. Together with the degree bound on passive supervertices, we have the lemma below.

Lemma A.2. *The total number of edges in $\bar{G}$ incident to passive supervertices is at most $120\frac{\gamma}{\delta}m$.*

This lemma, together with the termination condition that at least $\frac{1}{20}$ fraction of the edges in $\overline{G}$ are incident to passive supervertices, gives that $\overline{G}$ has $O(m_G \ln m_G/\delta)$ edges.

So far we have shown Part (1),(3),(4) of the correctness proof, and what remains is the termination of the outer loop.

For this purpose we will show that the number of edges in $\overline{G}$ decreases geometrically in every iteration of the while loop, while the number of edges in $\overline{G}$ incident to passive supervertices does not decrease as a supervertex, once it is passive, is always removed from H and, thus, it is never contracted into a new supervertex. Thus, the outer loop terminates after $O(\log n)$ iterations.

To show the reduction on edges in $\overline{G}$ we first need a new definition: The edges *cut* from H are those edges that (a) either are incident to removed passive supervertices or (b) that are returned by the inner loop as edges on a low conductance cut and that then are removed from H. Lemma A.4, which is Lemma 17 in [12], bounds the number of edges removed during trimming, shaving, and scraping by the number of edges that were cut. For this we need, however, the following auxiliary lemma.

Lemma A.3. *If a cluster C has k edges leaving it in G and the core of C is scraped, then $\mathbf{vol}_{\overline{G}}(C) \leq 4k$.*

Lemma A.4. *If the total number of edges cut during an iteration of the outer loop is c, the total number of edges lost from all clusters due to trimming, shaving and scraping during this iteration is $6c$.*

We use this lemma together with the fact that each cut found in the middle procedure has low conductance to conclude that the number of edges in $\overline{G}$ is reduced by a constant factor in each iteration of the outer loop.

Lemma A.5. *In each except for the last iteration of the outer loop, i.e. the repeat loop, the number of edges in the graph $\overline{G}$ is decreased by a factor of at least 7/10.*

The above lemma gives the termination of the outer loop, and completes the analysis of the K-T framework. We summarize the results of this subsection in the following theorem.

Theorem A.1. *Given a simple graph $G = (V, E)$ with minimum degree δ the presented mincut algorithm computes a multi-graph $\bar{G} = (\bar{V}, \bar{E})$ with $O(m_G \ln m_G/\delta)$ edges such that all non-trivial mincuts of G are non-trivial mincuts in $\bar{G}$.*

B Analysis of flow algorithm from Section 3

B.1 Analysis of *Unit-Flow*. Recall the *Unit-Flow* procudure (Algorithm 3.1) is a fairly straightforward implementation of the push-relabel framework, with some notable design decisions as follows:

- We explicitly maintain upperbounds on the supply remaining at a vertex, i.e. $f(v) \leq wd(v)$ when we push to v. We assume this holds at the start, i.e. the input $\Delta(v) \leq wd(v)$ for all v.
- We cap the labels at h. If a vertex has label $h-1$, and we relabel it to h, the vertex never becomes active from then on.
- The active vertices in Q are in non-decreasing order with respect to their labels, and each time we need to get an active vertex from Q, we get the first vertex.

Note that the assertion in *Push*(v, u) is the reason we always use the active vertex v with the smallest label. If *Push*(v, u) can be applied, but $f(u) \geq wd(u)$, we know $l(v) = l(u) + 1$, so $l(u) < h$, and u has positive excess as $w \geq 2$, then u is active, which contradicts v being the active vertex with the smallest label. The applicability conditions and the assertion guarantee that we can push $\psi \geq 1$ unit of supply from v to u.

Upon termination, we have a pre-flow f, and labels l on vertices. We make the following observations.

OBSERVATION B.1. *During the execution of* Unit-Flow, *we have*

(1) If v is active at any point, the final label of v cannot be 0. The reason is that v will remain active until either $l(v)$ is increased to h, or its excess is pushed out of v, which is applicable only when $l(v)$ is larger than 0 at the time of the push.

(2) Each vertex v is a sink that can absorb up to $d(v)$ units of supply, so we call the $f(v) - \text{ex}(v) = \min(f(v), d(v))$ units of supply remaining at v the absorbed *supply. The amount of absorbed supply at v is between $[0, d(v)]$, and is non-decreasing. Thus any time after the point that v first becomes active, the amount of absorbed supply is $d(v)$. In particular any time the algorithm relabels v, there have been $d(v)$ units of supply absorbed by v.*

Upon termination of Unit-Flow *procedure, we have*

(3) For any edge $\{v, u\} \in E$, if the labels of the two endpoints differ by more than 1, say $l(v) - l(u) > 1$, then arc (v, u) is saturated. This follows directly from a standard property of the push-relabel framework, where $r_f(v, u) > 0$ implies $l(v) \leq l(u) + 1$.

Although *Unit-Flow* may terminate with $\text{ex}(v) > 0$ for some v, we know all such vertices must have label h, as the algorithm only stops trying to route v's excess supply to sinks when v reaches level h. Thus we have the following lemma.

LEMMA B.1. *Upon termination of* Unit-Flow *with input (G, Δ, U, h, w), assuming $\Delta(v) \leq wd(v)$ for all v, the pre-flow and labels satisfy*

(a) If $l(v) = h$, $wd(v) \geq f(v) \geq d(v)$;

(b) If $h - 1 \geq l(v) \geq 1$, $wd(v) \geq f(v) = d(v)$;

(c) If $l(v) = 0$, $f(v) \leq d(v)$.

Proof. By Observation B.1.(2), any vertex with label larger than 0 must have $f(v) \geq d(v)$. The algorithm terminates when there is no active vertices, i.e. $\text{ex}(v) > 0 \implies l(v) = h$, so all vertices with label below h must have $f(v) \leq d(v)$. Moreover, $f(v) \leq wd(v)$ since at the beginning $f(v) = \Delta(v) \leq wd(v)$, and the push operations explicitly enforces $f(v) \leq wd(v)$ when pushing supply to v.

Now we can prove the main result about *Unit-Flow.*
Proof of Theorem 3.1

Proof. We use the labels at the end of *Unit-Flow* to divide the vertices into groups

$$B_i = \{v | l(v) = i\}$$

If $B_h = \emptyset$, no vertex has positive excess, so all $|\Delta(\cdot)|$ units of supply are absorbed by sinks, and we end up with case (1).

If $B_h \neq \emptyset$, but $B_0 = \emptyset$, by Lemma B.1 every vertex v has $f(v) \geq d(v)$, so we have $\sum_v d(v) = 2m$ units of supply absorbed by sinks, and we end up with case (2).

Case (3a): When both B_h and B_0 are non-empty, we compute the cut $(A, V \setminus A)$ as follows: Let $S_i = \cup_{j=i}^{h} B_j$ be the set of vertices with labels at least i. We sweep from h to 1, and let A be the first i such that $\Phi(S_i) \leq 20(\frac{\ln 2m}{h} + \frac{w}{U})$. The properties $\forall v \in A : wd(v) \geq f(v) \geq d(v)$, and $\forall v \in V \setminus A : f(v) \leq d(v)$ follow directly from $S_h \subseteq A \subseteq S_1$. We will show that there must exists some S_i satisfying the conductance bound.

For any i, an edge $\{v, u\}$ across the cut S_i, with $v \in S_i, u \in V \setminus S_i$, must be one of the two types:

1. In the residual network, the arc (v, u) has positive residual capacity $r_f(v, u) > 0$, so $l(v) \leq l(u) + 1$. But we also know $l(v) \geq i > l(u)$ as $v \in S_i, u \in V \setminus S_i$, so we must have $l(v) = i, l(u) = i - 1$.
2. In the residual network, if $r_f(v, u) = 0$, then (v, u) is a saturated arc sending U units of supply from S_i to $V \setminus S_i$.

Suppose there are $z_1(i)$ edges of the first type, and $z_2(i)$ edges of the second type. By the following region growing argument, we can show there exists some choice of $i = i^*$, such that

$$\text{(B.1)} \quad z_1(i^*) \leq \frac{10 \min(\mathbf{vol}(S_{i^*}), 2m - \mathbf{vol}(S_{i^*})) \ln m}{h}$$

If $\mathbf{vol}(S_{\lfloor h/2 \rfloor}) \leq m$, we start the region growing argument from $i = h$ down to $\lfloor h/2 \rfloor$. By contradiction, suppose $z_1(i) \geq \frac{10\mathbf{vol}(S_i) \ln m}{h}$ for all $h \geq i \geq \lfloor h/2 \rfloor$, which implies $\mathbf{vol}(S_i) \geq \mathbf{vol}(S_{i+1})(1 + \frac{10 \ln m}{h})$ for all $h > i \geq \lfloor h/2 \rfloor$. Since $\mathbf{vol}(S_h) = \mathbf{vol}(B_h) \geq 1$ and $h \geq \ln m$, we will have $\mathbf{vol}(S_{\lfloor h/2 \rfloor}) \geq (1 + \frac{10 \ln m}{h})^{h/2} \gg 2m$, which gives contradiction. The case where $\mathbf{vol}(S_{\lfloor h/2 \rfloor}) > m$ is symmetric, and we run the region growing argument from $i = 1$ up to $\lfloor h/2 \rfloor$ instead.

For any i, we can bound $z_2(i)$ as follows. Since the pre-flow pushes $z_2(i)U$ units of supply from S_i to $V \setminus S_i$ along the $s_2(i)$ saturated arcs, $z_2(i)U$ is at most $\sum_{v \in S_i} \Delta(v) + z_1(i)U$, i.e. the sum of the source supply in S_i and the supply pushed into S_i along the $z_1(i)$ eligible arcs. As $\Delta(v) \leq wd(v)$ for all v, we know

$$z_2(i) \leq \frac{w\mathbf{vol}(S_i)}{U} + z_1(i)$$

On the other hand, $z_2(i)U$ is at most $\sum_{v\in V\setminus S_i} f(v) + z_1(i)U$, as the $z_2(i)U$ units of supply pushed into $V \setminus S_i$ either remain at vertices in $V \setminus S_i$, or back to S_i along the reverse arcs of the $z_1(i)$ eligible arcs. Since any $v \in V \setminus S_i$ is not with label h, thus $f(v) \leq d(v)$, then we get

$$z_2(i) \leq \frac{\mathbf{vol}(V \setminus S_i)}{U} + z_1(i)$$

The two upperbounds of $z_2(i)$ together give

$$\text{(B.2)} \quad z_2(i) \leq \frac{w \min(\mathbf{vol}(S_i), 2m - \mathbf{vol}(S_i))}{U} + z_1(i)$$

We know there exists some i^* such that $z_1(i^*)$ is bounded by (B.1), together with (B.2), we have

$$z_1(i^*)+z_2(i^*) \leq \min(\mathbf{vol}(S_{i^*}), 2m-\mathbf{vol}(S_{i^*}))(\frac{20\ln m}{h}+\frac{w}{U})$$

thus $\Phi(S_{i^*}) \leq \frac{20\ln m}{h} + \frac{w}{U}$, which completes the proof.

Case (3b): The proof is basically the same as the above case, but with a more careful region growing argument. In particular, we want to show there exists some i^*, j^* such that $\min(\mathbf{vol}(S_{i^*}), 2m-\mathbf{vol}(S_{i^*})) \leq 2^{j^*}$ and

$$\text{(B.3)} \qquad z_1(i^*) \leq \frac{\mathbf{vol}(S_{i^*})(\log m + 1 - j^*)}{100 \log m'}$$

Assume $\mathbf{vol}(S_{\lfloor h/2 \rfloor}) \leq m$, and we run the region growing argument from $i = h$ down to $i = \lfloor h/2 \rfloor$. In this case $\mathbf{vol}(S_i) \leq m$. The other case is similar, and we just do the region growing argument from the other side.

Consider the groups $S_h, \ldots, S_{\lfloor h/2 \rfloor}$, if we put the $\Theta(\ln m' \ln\ln m')$ groups into levels $j = 1, \ldots, \log m$, such that a group i is in level j if $2^{j-1} \leq \mathbf{vol}(S_i) \leq 2^j$. There must be a level j that gets more than $200\frac{\log m'}{\log m+1-j}$ groups, as long as $h > c_2 \ln m' \ln\ln m'$ for some large constant c_2, since

$$\sum_{j=0}^{\log m} \frac{200 \log m'}{\log m + 1 - j} = 200 \log m'(1 + \frac{1}{2} + \cdots + \frac{1}{\log m + 1})$$
$$\leq 500 \ln m' \ln\ln m'$$

Suppose this is level j^*, and let i_a be the largest i with S_i in level j^*, and i_b be the smallest i with S_i in level j^*. We know $i_a - i_b \geq 200\frac{\log m'}{\log m+1-j^*}$, and $j^* - 1 \leq \log \mathbf{vol}(S_{i_a}) \leq \log \mathbf{vol}(S_{i_b}) \leq j^*$, thus there must be a $i^* \in [i_a, i_b]$ satisfying (B.3), since otherwise $\mathbf{vol}(S_{i_b}) \geq (1 + \frac{\log m+1-j^*}{\log m'})^{200\frac{\log m'}{\log m+1-j^*}} \gg 2\mathbf{vol}(S_{i_a})$. Everything else follow the same arguments as in the proof of case $(3a)$ above.

We proceed to prove the runtime of *Unit-Flow*. Recall we treat units of supply as distinct tokens, so a push operation of ψ units takes $O(\psi)$ work to maintain the marks.

Proof of Lemma 3.1

Proof. With a compact representation of Δ, the initialization of $f(v)$'s and Q takes time linear in $|\Delta(\cdot)|$. For the subsequent work, we will first charge the operations in each iteration of *Unit-Flow* to either a push or a relabel. Then we will in turn charge the work of pushes and relabels to the absorbed supply, so that each unit of absorbed supply gets charged $O(wh)$ work. This will prove the result, as there are at most $|\Delta(\cdot)|$ units of (absorbed) supply in total.

In each iteration of *Unit-Flow*, we look at the first element v of Q, which is an active vertex with the smallest label. Suppose $l(v) = i$ at that point. If the call to *Push/Relabel*(v) ends with a push of ψ units of supply, the iteration takes $O(\psi)$ total work and we charge the work of the iteration to that push operation. If the call to *Push/Relabel*(v) doesn't push, we charge the $O(1)$ work of the iteration to the relabel of $l(v)$ to $i + 1$. If there is no such relabel, i.e. i is the final value of $l(v)$, we know $i \neq 0$ by Observation B.1(1), then we charge the work to the final relabel of v. Since a relabel of v must be incurred when $d(v)$ consecutive calls to *Push/Relabel*(v) end with non-push, each relabel of v takes $O(d(v))$ work by our charging scheme above.

So far we have charged all the work to pushes and relabels, such that pushing ψ units of supply takes $O(\psi)$, and each relabel takes $O(d(v))$. We now charge the work of pushes and relabels to the absorbed supply. We consider the absorbed supply at v as the first up to $d(v)$ units of supply starting at or pushed into v, and these units never leave v.

By Observation B.1(2), each time we relabel v, there are $d(v)$ units of absorbed supply at v, so we charge the $O(d(v))$ work of the relabel to the absorbed supply, and each unit gets charged $O(1)$. A vertex v is relabeled at most h times, so each unit of absorbed supply is charged with $O(h)$ in total by all the relabels.

For the pushes, we consider the potential function

$$\Lambda = \sum_v \mathrm{ex}(v) l(v)$$

Each push operation of ψ units of supply decrease Λ by exactly ψ, since ψ units of excess supply is pushed from a vertex with label i to a vertex with label $i - 1$. Λ is always non-negative, and it only increases when we relabel some vertex v with $\mathrm{ex}(v) > 0$. When we relabel v, Λ is increased by $\mathrm{ex}(v)$. Since $\mathrm{ex}(v) \leq f(v) \leq wd(v)$, we can charge the increase of Λ to the absorbed supply at v, and each unit gets charged with $O(w)$. In total we can charge all pushes (via Λ) to absorbed supply, and each unit is charged with $O(wh)$.

If we need to compute the cut A as in case (3) of Theorem 3.1, the runtime is $O(\mathbf{vol}(S_1))$. Recall S_1 is the set of vertices with label at least 1, thus all with $d(v)$ units of absorbed supply, so $\mathbf{vol}(S_1)$ is at most $|\Delta(\cdot)|$.

B.2 Analysis of excess scaling procedure. Now we proceed to prove Lemma 3.2.

Proof. Consider the call to *Unit-Flow* in one iteration of the flow procedure with the unit being μ. The edge capacity used in *Unit-Flow* is U units, i.e. μU, and the total source supply to *Unit-Flow* is at most $\frac{2m}{\mu}$ units, so the runtime of *Unit-Flow* is $O(\frac{mh}{\mu})$. As μ decreases geometrically starting with F, the total edge capacity used through the procedure is $2UF$, and the total runtime will be $O(\frac{mh}{\mu})$ for the μ at termination.

The procedure will terminate either when each vertex v gets at most $d(v)$ supply, which must happen once μ drops to 1, or in some iteration j we have

$$\mathbf{vol}(A_j) \geq \frac{\tau 2m}{10\mu \ln 2\mu \ln\ln m}. \tag{B.4}$$

(When *Unit-Flow* finishes with case (1) or (2) of Theorem 3.1, $A_j = \emptyset$).

We need to argue (corrsponding to the two cases in this Lemma respectively)

1. If we don't terminate early due to Eqn. (B.4), then at least $(1-\tau)$ fraction of the total source supply is routed to sinks.

2. If we terminate due to Eqn. (B.4), we must have $\mathbf{vol}(A)$ being $\Omega(\frac{m}{F \ln m \ln\ln m})$, and the runtime in this case is $O(\mathbf{vol}(A) h \ln \frac{m}{\mathbf{vol}(A)} \ln\ln m)$, where A is the side of $(A_j, \bar{A}_j)$ with smaller volume. The conductance of the cut in this case follows from Theorem 3.1 case $(3a), (3b)$.

We first show case (1). By Theorem 3.1 with $w = 2$, the A_j we get from *Unit-Flow* satisfies $2d(v) \geq f(v) \geq d(v)$ for all $v \in A_j$, and $f(v) \geq d(v)$ for all $v \in V \setminus A_j$. So we have $\mathrm{ex}_j(A_j) \leq \mathbf{vol}(A_j) \leq \frac{2m}{\mu_j}$ where we use μ_j to denote the value of μ in the iteration of j. If we never have Eqn. (B.4), we know for all j

$$\mathrm{ex}_j(A_j)\mu_j \leq \mathbf{vol}(A_j)\mu_j \leq \frac{\tau 2m}{10 \ln 2\mu_j \ln\ln m}$$

and if we add up the excess removed from all iterations, we have

$$\sum_j \mathrm{ex}_j(A_j)\mu_j \leq \frac{\tau m}{5 \ln\ln m}\left(\sum_{j=0}^{\ln F} \frac{1}{j+1}\right) \leq \tau 2m$$

so the total supply remaining is at least $(1-\tau)2m$, and we have case (1). In this case when we terminate μ is at least 1, so the runtime is $O(mh)$.

For case (2), let j be the iteration when we get Ean. (B.4), and we look at the cut A_j returned. Let A be the smaller side of A_j and $\bar{A}_j$. The conductance of cut $(A_j, V \setminus A_j)$ follows from Theorem 3.1 with $w = 2$. We proceed to prove the runtime bound. We look at the two cases:

- $\mu_j \geq 2$ at termination: Since $\mathbf{vol}(A_j) \leq \frac{2m}{\mu_j}$, we have $\mu_j \leq \frac{2m}{\mathbf{vol}(A_j)}$, thus we can rewrite Eqn. (B.4) as
$$\mathbf{vol}(A_j) \geq \frac{\tau 2m}{10\mu \ln \frac{4m}{\mathbf{vol}(A_j)} \ln\ln m}.$$
The rumtime is $O(\frac{m}{\mu_j} h)$, which is also $O(\mathbf{vol}(A) h \ln \frac{m}{\mathbf{vol}(A)} \ln\ln m)$ (notice A_j is the smaller side of the cut when $\mu_j \geq 2$.)

- $\mu_j = 1$ at termination: If $\mathbf{vol}(A_j) \leq m$, A_j is still the smaller side, we have the same argument as above. When $\mathbf{vol}(A_j) \geq m$, either we have $\mathbf{vol}(A_j) \geq (1-\tau)2m$, which means at least $(1-\tau)$ fraction of total supply routed to sinks (i.e. case (1) of this Lemma), or we have $\mathbf{vol}(A_j)$,$\mathbf{vol}(V \setminus A_j)$ both $\Theta(m)$, since τ is a constant. The runtime is $O(mh)$, which is $O(\mathbf{vol}(A)h)$.

In both cases, the running time is $O(\mathbf{vol}(A) h \ln \frac{m}{\mathbf{vol}(A)} \ln\ln m)$.

It remains to show $\mathbf{vol}(A)$ is $\Omega(\frac{m}{F \log m})$. From the above discussion, we know that if we end with case (2) of the lemma, then either $\mathbf{vol}(A_j) \leq m$, or $\mathbf{vol}(\bar{A}_j) \geq \frac{\tau}{1-\tau}\mathbf{vol}(A_j)$. Thus the termination condition Eqn (B.4) implies $\mathbf{vol}(A)$ is $\Omega(\frac{m}{F \ln m \ln\ln m})$.

C Details and analysis of the inner procedure

In this section we will describe our inner procedure (Algorithm C.1), which largely follows the same approach as [12].

Recall when the inner procedure is invoked, we have a connected component C of H such that C is certified to be s-strong for some $s \in [s_0, m_C]$. Through the rest of the section, we work completely inside C, and the volume, degree, cut-size are all internal to C when we omit the subscript.

Recall the K-T framework removes passive supervertices from H, and keeps H trimmed through the algorithm. Thus in the connected component C, any regular vertex v has $d(v) \geq \frac{2}{5}\delta$, and any supervertex has degree at least $\frac{2}{5}c_1\delta\gamma$ where $\delta = \Theta(\ln m)$ is the parameter in the definition of a passive supervertex. Moreover, no two vertices in C have more than δ parallel edges between them, since such pair of vertices will be contracted.

As discussed in Section 4, we need to prove Theorem 5.1, and we will follow the intuitions outlined in Section 5.

As suggested earlier, we will construct various flow problems aiming to exploit the existence of a cut S of cut-size at most δ, and use the flow-based algorithms from Section 3 on the constructed problems. The edge capacity parameter is crucial if we want to use the cut as a bottleneck to route supply out of or into S. As specified in Lemma 3.2, when given parameter U, the actual edge capacities used by the algorithm is UF where $F = \max_v \frac{\Delta(v)}{d(v)}$, thus it is important for us to construct flow problems where the source function Δ has small F. Formally, our strategy to construct source function with small F is captured in the following definitions.

Definition C.1. *An* edge-bundle *is a set of edges sharing a common endpoint. We denote an edge-bundle by $(v, X(v))$, where v is the common endpoint that we call the* center *of the edge-bundle, and $X(v)$ is the multiset (as there are parallel edges) containing the other endpoints of the edges. A set of edge-bundles are disjoint if their underlying sets of edges are edge disjoint.*

Note that in a set of disjoint edge-bundles, a vertex v can still be the center of multiple edge-bundles, and we can also have parallel edges, the definition simply prevents the same edge from appearing in multiple edge-bundles.

Definition C.2. *Given a set Y of edge-bundles in $C = (V, E)$, the* expansion graph *associated with Y is the directed multigraph $G_Y = (V, E_Y)$, such that E_Y has a directed edge (v, u) for each $u \in X(v)$ and each $(v, X(v)) \in Y$. Namely, E_Y is the union of all edge-bundles in Y, with edges oriented away from the centers of the edge-bundles.*

Definition C.3. *A set of edge-bundles Y in C is* (α, Z)-sparse *if*

- *The edge-bundles in Y are edge disjoint.*
- *Each edge-bundle $(v, X(v)) \in Y$ has at least Z edges.*
- *For each vertex v, its in-degree in the associated expansion graph G_Y is at most $\frac{1}{\alpha}$ of its degree in C.*

Note if Y is (α, Z)-sparse, then any subset of Y is also (α, Z)-sparse.

Edge-bundles will be used to construct source functions for flow problems, and the motivation of (α, Z)-sparse set of edge-bundles is that if we put supply on the centers of the edge-bundles in the set, and push out uniformly using the edges in the edge-bundles, the amount of supply received by any node will not be too large comparing to its degree.

More precisely, we call an *initial spread-out* of σ supply over the edge-bundle $(v, X(v))$ as the operation of starting with σ supply on v, and pushing $\frac{\sigma}{|X(v)|}$ supply along each edge in the edge-bundle to vertices in $X(v)$. Formally, given edge-bundle $(v, X(v))$ and σ, we define

$$\Delta_{(v,X(v)),\sigma}(u) = \frac{\sigma \cdot \#(u, X(v))}{|X(v)|}$$

where $\#(u, X(v))$ is the number of times u appears in $X(v)$. That is, $\Delta_{(v,X(v)),\sigma}(u)$ is the supply ending at u if we start with σ supply at the center v of the edge-bundle $(v, X(v))$, and then push out all the σ supply at v evenly along the edges in the edge-bundle. We extend the definition to a set Y of edge-bundles:

$$\Delta_{Y,\sigma}(u) = \sum_{(v,X(v))\in Y} \Delta_{(v,X(v)),\sigma}(u)$$

i.e. $\Delta_{Y,\sigma}(u)$ is the amount of supply ending at u, if we carry out simultaneously a initial spread-out of σ supply over each edge-bundle in Y. It is clear that the total amount of supply is $|\Delta_{Y,\sigma}(\cdot)| = |Y|\sigma$, where $|Y|$ is the number of edge-bundles in Y.

We will use the supply on vertices arising from initial spread-outs as the source function, and we consider flow problems defined below.

Definition C.4. *Given $\Delta : V \to \mathbb{Z}_{\geq 0}$ and β, we define a flow problem,* Flow-Problem(Δ, β)*, as follows. The source function is given by $\Delta(\cdot)$, all edges have capacity β, and each vertex v is a sink of capacity $d(v)$.*

Essentially we are taking a two-phase approach to spread supply from edge-bundle centers to the entire graph. The first phase being the initial spread-outs, where we have full control of the behavior, and the second phase being the flow routing, so we can still take advantage of the better conductance property of flow algorithms.

The flow algorithm we use in Section 3.1 and Section 3.2 will also allow us to associate each unit of supply with its source vertex as specified by $\Delta(\cdot)$. When the $\Delta(\cdot)$ we use arises from initial spread-outs over edge-bundles, we can further decompose the flow to associate each unit of supply with the original edge-bundle it started at, i.e. before the initial spread-out. Thus in step 3, we assume we know how much of the supply originating from each edge-bundle is routed to sinks.

Algorithm C.1. Inner Procedure

Input: Trimmed component C with m_C edges, and $s \in [s_0, m_C]$ such that C is s-strong.
Steps:

1. Choose a set Y of $\frac{5000 m_C}{s}$ edge-bundles that is $(\gamma, \frac{\delta}{10})$-sparse, and split Y into sets $Y_1, \dots, Y_{5000}$, each with $\frac{m_C}{s}$ edge-bundles.
2. In parallel (step by step) for all $i = 1, \dots, 5000$, solve Flow-Problem$(\Delta_{Y_i,2s}, \frac{s}{1000\delta})$ using Algorithm 3.2 in Section 3.2, with inputs graph C, source function $\Delta_{Y_i,2s}$, $\tau = 0.1$, $U = 100 \ln m_G$, and $h = 1000 \ln m_G \ln\ln m_G$. Terminate all problems if the Flow-Problem of any i terminates with a cut A as in case (2) of Lemma 3.2, stop the inner procedure with A.
3. Otherwise, the Flow-Problems for all i end with case (1) of Lemma 3.2, i.e. with at least $1.8m$ supply routed to sinks. For each Flow-Problem i, use the returned preflow f_i to find a subset $X_i \subseteq Y_i$ such that each edge-bundle in X_i has at least $1.6s$ of its $2s$ initial supply routed to sinks.
4. For each i, compute $g_i(\cdot)$ from $f_i(\cdot)$ as follows: First remove from each $f_i(v)$ the excess supply on vertices (i.e. $\max(f_i(v) - d(v), 0)$ supply on v), as well as the supply not originating from edge-bundles in X_i. Then scale the remaining supply at every vertex by $\frac{1}{200}$.
5. Let $\Delta_X(\cdot) \overset{\text{def}}{=} \sum_i g_i(\cdot)$. Run *Unit-Flow* in Section 3.1 with inputs $G = C$, source function Δ_X, $U = \frac{s}{20\delta}$, $h = 1000 \ln m_G \ln\ln m_G$, and $w = 25$. If the returned preflow routes at least $d(v)$ supply to every vertex v, stop and output that C is $0.6s$-strong. Otherwise, stop with the set A returned by *Unit-Flow*, and output that A is $0.6s$-strong.

We use the following definition to formally specify whether an edge-bundle is "inside" or "outside" a small cut.

Definition C.5. *An edge-bundle $(v, X(v))$ is s-captured if there is a cut S such that $\partial(S) \le \delta$, $s_0 \le \mathbf{vol}(S) \le s$, and $|X(v) \cap S| \ge \frac{3}{4}|X(v)|$, i.e. at least $\frac{3}{4}$ of the edges are between v and vertices in S. We say the edge-bundle is s-captured by S. (Note that v might or might not belong to S.) A s-*free *edge-bundle is one that is not s-captured.*

In Step 1 of the inner procedure, we pick a large set Y of edge-bundles that is $(\gamma, \frac{\delta}{10})$-sparse. This step is valid as we have the folloing lemma.

Lemma C.1. *For $s_0 \ge 1000\gamma\delta$, a trimmed component $C = (V, E)$ with $m = |E|$, and any $m \ge s \ge s_0$, we can construct a set of $\frac{5000m}{s}$ edge-bundles that is $(\gamma, \frac{\delta}{10})$-sparse. The construction takes $O(\frac{m\delta\gamma}{s})$.*

Proof. Let $Z = \frac{\delta}{10}$, in our construction, we say a super-vertex is *live* if it has at least γZ edges to live neighbors, and a regular vertex is *live* if it has at least Z edges to live neighbors. Vertices are *dead* if not *live*. We will implicitly consider a graph C' on live vertices. As C being a trimmed component, we know at the start all regular vertices have degree at least $4Z$, and all super-vertices have degree at least $4\gamma Z$, so we can make all vertices live at the start, and $C' = C$. As $\delta \gg \gamma$ in our setting, for simplicity we assume integrality of $\frac{d_{C'}(v)}{\gamma}$ for any live vertex v, as $d_{C'}(v) \ge Z = \frac{\delta}{10}$.

Now we describe how we construct the set of edge-bundles Y as follows:

1. Choose an arbitrary live super-vertex, if no live super-vertex exists, choose a live regular vertex. Call the chosen vertex v.
2. Construct an edge-bundle centered at v by picking Z incident edges of v in C', subject to the constraint that for each live neighbor u of v, we pick at most $\min(d_{C'}(v,u), \frac{d_{C'}(u)}{\gamma})$ parallel edges between u and v, where $d_{C'}(v,u)$ is the number of edges between v and u in C'. Add the edge-bundle to Y.
3. Remove edges from C' as follows
   (a) For each edge $\{v, u\}$ added to the edge-bundle above, remove that edge and an additional $\gamma - 1$ incident edges of u from C'.
   (b) Recursively remove from C' the dead vertices and all their incident edges.
4. Repeat the process from Step 1 until we have $\frac{5000m}{s}$ edge-bundles in Y.

First we show that Step 2 is always feasible, i.e. we can obtain such an edge-bundle with a live vertex v. As all vertices in C' are live, if v is a super-vertex, the number of edges we can pick is

$$\sum_u \min(d_{C'}(v,u), \frac{d_{C'}(u)}{\gamma}) \ge \sum_u \frac{d_{C'}(v,u)}{\gamma} \ge \frac{d_{C'}(v)}{\gamma} \ge Z$$

If v is a regular vertex, C' must have no super-vertex at that point, so there are no parallel edges in C'. In this case, we can add any incident edge (v, u) to the edge-bundle of v, and v has at least Z incident edges in C'.

The condition we enforce in Step 2 guarantees that we can always carry out Step $3(a)$, i.e. there will be enough edges to remove. If we have added k edge-bundles to Y, the total number of edges we removed in Step $3(a)$ is $kZ\gamma$. To bound the number of edges removed in Step $3(b)$, assume that every removal in Step 2 and Step 3(a) places one token on the other endpoint of the removed edge. Thus, a total of $kZ\gamma$ tokens are placed on nodes. We will show that we can place one token on each edge removed during Step 3(b), which bounds the number of edges removed in Step 3(b) by $kZ\gamma$. Whenever a dead vertex v is removed, it places one token on each removed adjacent edge and it gives one token to the other endpoints of this edge. It remains to show that v has a sufficient number of tokens to do so. We show this by showing by induction the more general claim that at each point in time the number of tokens placed on a vertex correponds to the number of edges the vertex has already lost. This then guarantees that each dead vertex that is removed has a sufficient number of tokens to give to its removed edges and its neighbors, as such a vertex has lost at least 3/4-th of its adjacent edges. The claim certainly holds before and when the first dead vertex is removed as it received a token for all its previously removed edges. Next consider the removal of the i-th dead vertex v and assume by induction that right before the removal v has at least $3d_C(v)/4$ many tokens. As v has lost at least 3/4-th of its edges, the removal of v removes at most $d_C(v)/4$ many edges. We use $d_C(v)/4$ many of v's tokens and give them to the removed edges and another $d_C(v)/4$ many tokens and give them to the other endpoints of the removed edges. Thus, the induction invariant also holds after the removal of the i-th dead vertex. This bounds the total number of edge removed in Step $3(b)$ by the total number of edges removed in Step 2 and $3(a)$, and thus after adding k edge-bundles to Y, we have removed $2kZ\gamma$ edges from C'.

As long as C' is not empty, it guarantees a live vertex, and thus an edge-bundle to add. We showed that the total number of edges removed from C' after constructing k edge-bundles is $2kZ\gamma$, thus as long as $2kZ\gamma \leq m$, we must have edges remaining in C'. This implies that we can have at least $\frac{m}{Z\gamma} \geq \frac{5m}{\delta\gamma}$ edge-bundles, so as long as $s \geq s_0 \geq 1000\delta\gamma$, we can have $\frac{5000m}{s}$ edge-bundles.

The set of edge-bundles is clearly $(\gamma, \frac{\delta}{10})$-sparse by our construction as for each edge that we added to an edge-bundle, and that will become an in-edge for a vertex v we removed $\gamma - 1$ edges incident to v. As to the runtime, since we implicitly keep C', the work is linear in the total number of edges removed from C, which is $2Z\gamma$ per edge-bundle, thus $O(\frac{m\delta\gamma}{s})$ in total.

First we show that if the procedure terminates at Step 2, we get a cut as specified in case (1) of Theorem 5.1.

LEMMA C.2. *If the inner procedure stops at Step 2, we have a set A with $\mathbf{vol}(A) \leq m$, and $\Phi(A) \leq \frac{(\log m+1-\lceil \log \mathbf{vol}(A) \rceil)}{20 \log m_G}$ in time $O(\mathbf{vol}(A) \ln \frac{m}{\mathbf{vol}(A)} \ln m_G \ln \ln^2 m_G)$.*

Proof. If the first i that the excess scaling flow algorithm terminates with case (2) of Lemma 3.2, let A be the smaller side of the cut returned. We know A has the desired conductance, as $|\Delta_{Y_i,2s}(\cdot)| = \frac{m}{s} \cdot 2s = 2m$, $h = 1000 \ln m_G \ln \ln m_G$ and $U = 100 \ln m_G$. As to the runtime, since we run all $O(1)$ flow problems in parallel, the time we spend before we terminate with A is $O(\mathbf{vol}(A) \ln \frac{m}{\mathbf{vol}(A)} \ln m_G \ln \ln^2 m_G)$ by Lemma 3.2. Furthermore, Lemma 3.2 guarantees that $\mathbf{vol}(A)$ is $\Omega(\frac{m}{F \ln m \ln \ln m})$, where $F = \max_v \frac{\Delta_{Y_i,2s}(v)}{2d(v)}$.

We now upper-bound the value of F in the Flow-Problems associated with the Y_i's. Since each Y_i is $(\gamma, \frac{\delta}{10})$-sparse, we know $\Delta_{Y_i,2s}(v) \leq \frac{2s}{\delta/10} \frac{d(v)}{\gamma}$ for all v by Definition C.3 and the construction of $\Delta_{Y_i,2s}$ from initial spread-outs. Thus $F \leq \frac{10s}{\delta\gamma}$, which implies $\mathbf{vol}(A)$ is $\Omega(\frac{m\delta\gamma}{s \ln m \ln \ln m})$. To find Y in the first step of the inner procedure, we spend time $O(\frac{m\delta\gamma}{s})$, which is $O(\mathbf{vol}(A) \ln m \ln \ln m)$. Thus the total runtime is $O(\mathbf{vol}(A) \ln \frac{m}{\mathbf{vol}(A)} \ln m_G \ln \ln^2 m_G)$ if the inner procedure ends in Step 2.

Now we formalize the intuition that if the source function has large enough initial source supply trapped inside the small cut, we get a very infeasible flow problem.

LEMMA C.3. *Given an s-captured edge-bundle $(v, X(v))$, we can send at most $1.6s$ supply to sinks in* Flow-Problem$(\Delta_{(v,X(v)),2s}, \frac{s}{1000\delta})$.

Proof. The flow problem we consider is with source function resulting from an initial spread-out of $2s$ supply over an edge-bundle $(v, X(v))$, which is s-captured by some set S. We know at least $\frac{3}{4}$ of the edges go to vertices in S, so after the initial spread-out, the total source supply at vertices outside S is at most $\frac{s}{2}$. As vertices in S have total sink capacity $\mathbf{vol}(S)$, which is at most s, the total amount of supply that can be routed to sinks in S is at most s. Furthermore, at most $\frac{s}{1000\delta}\delta$ supply can be pushed out of S, since the cut has size at most δ, and edges have capacity $\frac{s}{1000\delta}$. Even if all the $\frac{s}{2} + \frac{s}{1000} \leq 0.6s$ supply not in S is routed to sinks eventually, we have at most $1.6s$ supply routed to sinks in total.

Given the above lemma, if the Flow-Problems associated with all Y_i's successfully route most of the supply to sinks, we know many of the edge-bundles we start with are not s-captured.

Lemma C.4. *If $\gamma > 10^7 \ln m_G$, Step 3 of inner procedure will have at least $\frac{m}{10s}$ edge-bundles in each X_i, and all the edge-bundles in X_i are s-free.*

Proof. Lemma C.3 states that if an edge-bundle is s-captured, after an initial spread-out of $2s$ supply over the edge-bundle, at most $1.6s$ of the $2s$ supply can be subsequently routed to sinks, as long as the flow respects the edge capacity of $\frac{s}{1000\delta}$ on each edge. By Lemma 3.2, we know the edge capacity used in the excess scaling flow algorithm is $200F \ln m_G$, where by our calculation in Lemma C.2 we have $F \leq \frac{10s}{\delta\gamma}$. Thus the preflow f_i respects the edge capacity of $\frac{2000s \ln m_G}{\delta\gamma}$, which is less than $\frac{s}{1000\delta}$ when $\gamma > 10^7 \ln m_G$. By Lemma C.3, any s-captured edge-bundle in Y_i has at most $1.6s$ of its initial supply routed to sinks, so we know all edge-bundles in X_i are s-free.

To bound the size of X_i, note that we have $2s$ supply starting with each of the $\frac{m}{s}$ edge-bundles, thus if less than $\frac{m}{10s}$ of them have more than $1.6s$ rounted to sinks, we can have at most $\frac{m}{10s}2s + \frac{9m}{10s}1.6s < 1.8m$ total supply routed to sinks. Since at least $1.8m$ supply is routed to sinks, we must have at least $\frac{m}{10s}$ edge-bundles in each X_i.

If we get to Step 3 of the inner procedure, we have spent $O(mh)$ on all the flow problems in the earlier step, so we need to make progress by certifying that a subset of volume $\Omega(m)$ is $0.6s$-strong. We now formalize the strategy we outlined in the second half of Section 5.

Intuitively, we want to continue with the pre-flows we have from Step 2, since we know from these preflows that we can spread out the supply of all edge-bundles in X. However, if we simply start from scratch on edge-bundles in X, we may end up with some small cut, because flow routing is not a linear operator. The procedure we carry out in Step 4 is mainly to get around the non-linearity of flow routing, so we can essentially keep the work done in the earlier step on spreading out the supply of edge-bundles in X.

In Step 4, we get preflow g_i from the preflow f_i for each i, and the union $\sum_i g_i$ is also a preflow. This preflow must be source-feasible with respect to a implicit source function (i.e. if we reverse the preflow $\sum_i g_i$) which we call Δ_0 and that we only use for the analysis and do not need to compute. It is different from the Δ_X in Step 5: If we start with source function Δ_0, and route according to $\sum_i g_i$, we would have $\Delta_X(v)$ supply ending at v. Note in the actual algorithm, we only need to compute g_i's as the supply ending at vertices, i.e. $g_i(v)$'s, but not the actual routing, i.e. $g_i(u,v)$'s, as long as we know there is a valid routing that ends with the $g_i(v)$'s.

Lemma C.5. $50m \geq |\Delta_0(\cdot)| = |\Delta_X(\cdot)| \geq 4m.$

Proof. By construction $\Delta_0(\cdot)$ is the source function of a preflow, and $\Delta_X(\cdot)$ is the supply ending at vertices after the preflow. Thus $|\Delta_0(\cdot)| = |\Delta_X(\cdot)|$.

In Step 4, the amount of supply that g_i keeps from f_i is $\frac{1}{200}$ fraction of the non-excess supply originating from any edge-bundle in $X = \cup_{i=1}^{5000} X_i$. As any edge-bundle in X has at least $1.6s$ supply routed to sinks, and we have at least $\frac{5000m}{10s}$ edge-bundles in X by Lemma C.4, in total we keep at least $\frac{1.6s}{200}\frac{5000m}{10s} = 4m$ supply. By construction, Δ_X has all this supply, i.e., $|\Delta_X(\cdot)| \geq 4m$.

The upperbound of $50m$ is because each g_i has at most $\frac{2m}{200}$ total supply by scaling f_i, and the 5000 groups in total make it at most $50m$ total supply in $\Delta_X(\cdot)$.

Now we define a flow problem Π, where the source function is Δ_0, each vertex v is a sink of capacity $d(v)$, and edges have capacity $U = \frac{s}{10\delta}$. We first analyse Π and then use it to analyze Step 5 in the subsequent lemma.

Lemma C.6. *For any feasible preflow of* Π, *the set B of vertices with their sink capacities saturated, is* $0.6s$*-strong.*

Proof. Consider any cut S such that $\partial(S) \leq \delta$, $s_0 \leq \mathbf{vol}(S) \leq s$. We will bound the total amount of supply that can end in S for any feasible preflow of Π.

We first look at the amount of supply that starts in S. The preflow f_i starts with source function $\Delta_{Y_i,2s}$, so if we examine our construction of g_i from f_i, we can mimic the changes on $\Delta_{Y_i,2s}$ to obtain the source function of g_i. Thus, the source function Δ_0 can be obtained equivalently as follows: (i) We start with $\frac{2s}{200}$ supply (corresponding to the scaling) at the center of each edge-bundle in X (corresponding to only keeping supply orginating from edge-bundles in X); (ii) carry out the initial spread-outs; (iii) and then remove some supply (corresponding to the removal of excess supply in $f_i(\cdot)$). Then it is clear we can bound the amount of supply that Δ_0 has in S by the amount of supply that would have been in S without the Step (iii).

Since all edge-bundles in X are s-free, if any edge-bundle has its center v in S, at least $\frac{1}{4}$ of its $\frac{\delta}{10}$ edges cross $(S, \bar{S})$. As the cut-size is δ, among all edge-bundles in X, at most 40 of them have their centers in S, which means at most $\frac{2s}{200} \cdot 40 = 0.4s$ supply can be in S before all the initial spread-outs. Since X is a subset of $(\gamma, \frac{\delta}{10})$-sparse set Y, X is also $(\gamma, \frac{\delta}{10})$-sparse. Thus the initial spread-outs push at most $\frac{2s}{200}/(\frac{\delta}{10}) = \frac{s}{10\delta}$ supply along each edge. Thus, as the cut-size of S is δ, at most an additional $\frac{s}{10\delta} \cdot \delta = \frac{s}{10}$ supply can end in S after the initial spread-outs. Thus in total, the source function Δ_0 can have at most $0.4s + 0.1s = 0.5s$ supply starting in S.

Subsequently any valid preflow pushes at most $\frac{s}{10\delta}$ supply along each edge due to the edge capacity constraints in Π, so an additional $\frac{s}{10}$ supply can be routed into S by the preflow. In total that means at most $0.6s$ supply can end in any such set S.

Now consider B, the set of all v that receives at least $d(v)$ supply. We must have $\mathbf{vol}(B \cap S) \leq 0.6s$ for any set S such that $\partial(S) \leq \delta$, $s_0 \leq \mathbf{vol}(S) \leq s$. This is enough to certify that B is $0.6s$-strong, since B is already s-strong as a subcomponent of C.

This is equivalent to the definition of B being $0.6s$-strong, as we are working inside a connected component C of H, so $\mathbf{ivol}_H(B \cap S, B) \leq \mathbf{vol}_C(B \cap S)$.

We now finish the proof of Theorem 5.1 by showing the following lemma.

Lemma C.7. *Step 5 will in time $O(m \ln m_G \ln \ln m_G)$ either certify that the entire component C is $0.6s$-strong, i.e. Case (3) of Theorem 5.1, or find a subset A with*

$$\Phi(A) \leq \frac{(\log m + 1 - \lceil \min(\log \mathbf{vol}(A), \log \mathbf{vol}(V \setminus A)) \rceil)}{20 \log m_G}.$$

In the latter case A has volume $\Omega(m)$, and is certified to be $0.6s$-strong, i.e., Case (2) of Theorem 5.1.

Proof. In Step 5 of the inner procedure we run *Unit-Flow* of Section 3.1 with inputs $G = C$, source function Δ_X, $U = \frac{s}{20\delta}$, $h = 1000 \ln m_G \ln \ln m_G$, and $w = 25$. Note that it fulfills the assumptions on inputs of *Unit-Flow* in Theorem 3.1 as $h \gg \ln m$, $w \geq 2$ and, by the construction of Δ_X, which removes all excess supply from all preflows f_i, it holds that $\Delta_X(v) \leq \frac{d(v)}{200} 5000 = 25d(v) = wd(v)$ for all v. Let f be the pre-flow returned by the *Unit-Flow* invocation.

Recall $\sum_i g_i$ is source-feasible with respect to the source function $\Delta_0(\cdot)$, and by routing according to $\sum_i g_i$, the supply ending at each vertex v is $\Delta_X(v)$. Essentially f resumes the routing by having $\Delta_X(\cdot)$ as source-function, so we can piece $\sum_i g_i$ and f together, and obtain a preflow f^* that is source-feasible with respect to $\Delta_0(\cdot)$.

To show f^* is a feasible preflow for the flow problem Π, we need to further show f^* respects the $\frac{s}{10\delta}$ edge capacity of Π. From Step 2 of the inner procedure, we have each preflow f_i using at most $\frac{s}{1000\delta}$ edge capacity, thus by construction $\sum_i g_i$ uses edge capacity of at most $\frac{s}{1000\delta} \frac{5000}{200} = \frac{s}{40\delta}$. As in the *Unit-Flow* invocation we use edge capacity $U = \frac{s}{20\delta}$, the preflow f^*, as a union of $\sum_i g_i$ and f, routes at most $\frac{3s}{40\delta}$ supply on each edge, and is thus a feasible preflow of Π.

As f^* is a valid preflow of Π, we know from Lemma C.6 the set B, containing all vertices v receiving at least $d(v)$ supply, is $0.6s$-strong, and so is any subset of B. Since f^* appends f after $\sum_i g_i$, the supply ending at vertices is given by $f(\cdot)$, so $B = \{v | f(v) \geq d(v)\}$.

In Lemma C.5 we showed that the total supply $|\Delta_X(\cdot)|$ for Step 5 is at least $4m$. As all the vertices can absorb only $2m$ supply in total, our invocation of *Unit-Flow* won't end with case (1) of Theorem 3.1. If f returned by *Unit-Flow* fulfills Case (2) of Theorem 3.1, i.e. all vertices get at least $d(v)$ supply, we are guaranteed that C is $0.6s$-strong.

On the other hand if *Unit-Flow* returns a set A as in Case (4) of Theorem 3.1, we show (a) $\Phi(A) \leq \frac{(\log m + 1 - \lceil \min(\log \mathbf{vol}(A), \log \mathbf{vol}(V \setminus A)) \rceil)}{20 \log m_G}$, (b) A is certified to be $0.6s$-strong, and (c) $\mathbf{vol}(A)$ is $\Omega(m)$. This implies that A satisfies the conditions of Case (2) of Theorem 5.1.

(a) The property of conductance follows directly from Case (4) of Theorem 3.1: As we use $h = 1000 \ln m_G \ln \ln m_G, w = 25, U = \frac{s}{20\delta} \geq \frac{s_0}{20\delta} \geq \gamma \geq 10^7 \ln m_G$, Case (4) gives the desired conductance bound. (b) We know any vertex $v \in A$ receives at least $d(v)$ supply, so $A \subseteq B$ is $0.6s$-strong. (c) As any vertex $v \notin A$ receives at most $d(v)$ supply, any vertex $v \in A$ receives at most $50d(v)$ supply, and since there is at least $4m$ total supply, we must have $50\mathbf{vol}(A) + (2m - \mathbf{vol}(A)) \geq 4m$, which implies that $\mathbf{vol}(A) \geq \frac{2}{49}m = \Omega(m)$. Thus A satisfies all the conditions and the proof of the lemma is complete.

The runtime of case (2) and (3) of Theorem 5.1 is $O(m \ln m_G \ln \ln m_G)$, as that's the total runnning time of the steps involved.