Expander Decomposition and Pruning: Faster, Stronger, and Simpler.

Thatchaphol Saranurak [*] Di Wang [†]

October 31, 2018

Abstract

We study the problem of graph clustering where the goal is to partition a graph into clusters, i.e. disjoint subsets of vertices, such that each cluster is well connected internally while sparsely connected to the rest of the graph. In particular, we use a natural bicriteria notion motivated by Kannan, Vempala, and Vetta [KVV00] which we refer to as *expander decomposition*. Expander decomposition has become one of the building blocks in the design of fast graph algorithms, most notably in the nearly linear time Laplacian solver by Spielman and Teng [ST04], and it also has wide applications in practice.

We design algorithm for the parametrized version of expander decomposition, where given a graph G of m edges and a parameter ϕ, our algorithm finds a partition of the vertices into clusters such that each cluster induces a subgraph of conductance at least ϕ (i.e. a ϕ expander), and only a $\tilde{O}(\phi)$ fraction of the edges in G have endpoints across different clusters. Our algorithm runs in $\tilde{O}(m/\phi)$ time, and is the first nearly linear time algorithm when ϕ is at least $1/\log^{O(1)} m$, which is the case in most practical settings and theoretical applications. Previous results either take $\Omega(m^{1+o(1)})$ time (e.g. [NS17, Wul17]), or attain nearly linear time but with a weaker expansion guarantee where each output cluster is guaranteed to be contained inside some unknown ϕ expander (e.g. [ST13, ACL06]). Our result achieve both nearly linear running time and the strong expander guarantee for clusters. Moreover, a main technique we develop for our result can be applied to obtain a much better *expander pruning* algorithm, which is the key tool for maintaining an expander decomposition on dynamic graphs. Finally, we note that our algorithm is developed from first principles based on relatively simple and basic techniques, thus making it very likely to be practical.

[*]Toyota Technological Institute of Chicago, saranurak@ttic.edu. Work mostly done while was in KTH Royal Institute of Technology

[†]Georgia Institute of Technology, wangd@eecs.berkeley.edu.

1 Introduction

Graph clustering algorithms are extensively studied and have wide practical applications such as unsupervised learning, community detection, and image segmentation (see e.g. [For10, Sch07, SM00]). A natural bicriteria notion for graph clustering introduced by Kannan Vempala and Vetta [KVV00], which we refer to as *Expander decomposition*, is to decompose a graph into *clusters* such that each cluster is richly intraconnected and sparsely connected to the rest of the graph. More formally, given a graph $G = (V, E)$ we aim to find a partitioning of V into $V_1, \ldots, V_k$ for some k, such that the total number of edges across different clusters is small while the *conductance* of each cluster as an induced subgraph is large. This bicriteria measure is advantageous over other popular measures such as min diameter decomposition, k-center, and k-median since there are simple examples where these measures fail to capture the natural clustering. Moreover, expander decomposition has seen great applications in algorithm design including graph sketching/sparsification [ACK+16, JS18, CGP+18], undirected/directed Laplacian solvers [ST04, CKP+17], max flow algorithms [KLOS14], approximation algorithms for unique game [Tre05], and dynamic minimum spanning forest algorithms [NS17, Wul17, NSW17]. An efficient algorithm to compute expander decomposition is crucial for all these applications (See Section 4.2 for a more detailed discussion about applications). With the abundance of massive graphs, it is crucial to design algorithms with running time at most nearly linear in the size of the graph, and thus nearly linear time expander decomposition methods are of great interest both in theory and in practice.

To continue the discussion, we need to introduce some notations. For an undirected graph $G = (V, E)$, we denote $\deg(v)$ as the number of edges incident to $v \in V$, and $\mathbf{vol}(C) = \sum_{v \in C} \deg(v)$ as the *volume* of $C \subseteq V$. We use subscripts to indicate what graph we are working with, while we omit the subscripts when the graph is clear from context. A *cut* is treated as a subset $S \subset V$, or a partition $(S, \overline{S})$ where $\overline{S} = V \setminus S$. For any subsets $S, T \subset V$, we denote $E(S, T) = \{\{u, v\} \in E \mid u \in S, v \in T\}$ as the set of edges between S and T. The *cut-size* of a cut S is $\delta(S) = |E(S, \overline{S})|$. The *conductance* of a cut S in G is $\Phi_G(S) = \frac{\delta(S)}{\min(\mathbf{vol}_G(S), \mathbf{vol}_G(V \setminus S))}$. Unless otherwise noted, when speaking of the conductance of a cut S, we assume S to be the side of smaller volume. The conductance of a graph G is $\Phi_G = \min_{S \subset V} \Phi_G(S)$. If G is a singleton, we define $\Phi_G = 1$. Let $G[S]$ be the subgraph induced by $S \subset V$, and we denote $G\{S\}$ as the induced subgraph $G[S]$ but with self-loops added to vertices so that any vertex in S has the same degree as its degree in G. Observe that for any $S \subset V$, $\Phi_{G[S]} \geq \Phi_{G\{S\}}$. We say a graph G is a ϕ *expander* if $\Phi_G \geq \phi$, and we call a partition $V_1, \ldots, V_k$ of V a ϕ expander decomposition if $\min_i \Phi_{G[V_i]} \geq \phi$.

We first note that for any graph, there always *exists* an expander decomposition with the following guarantee:

Observation 1.1 (Ideal expander decomposition)**.** *Given any graph $G = (V, E)$ with m edges and a parameter $\phi \in (0, 1)$, there exists a partition $V_1, \ldots, V_k$ of V for some k such that $\min_i \Phi_{G[V_i]} \geq \phi$ and $\sum_i \delta(V_i) = O(\phi m \log n)$.*

The trade off in Observation 1.1 is tight. As observed in [AALG18], in a hypercube after deleting a small constant fraction of edges, some remaining components will have conductance $O(1/\log n)$. There is an simple and algorithmic argument for Observation 1.1 as follows. Given a graph G, we find the cut S with the smallest conductance (or interchangeably referred to as the sparsest cut in our discussion) . If $\Phi_G(S) < \phi$, we cut along S, and recursively decompose $G[S]$ and $G[\overline{S}]$. Otherwise, we know $\Phi_G \geq \phi$ so we can output V as an expander cluster, and it is clear all the output clusters are ϕ expanders. The number of edges across different clusters follow from a simple charging argument, where each time we cut the graph, we charge the number of edges we cut to the edges remaining in the smaller side of the cut. Since the cuts have conductance at most ϕ, and any edge can be on the smaller side of a cut at most $O(\log m)$ times, we know the total number of edges across clusters is at most $O(\phi m \log m)$.

Since finding the sparsest cut is NP-hard, a natural way to turn the above argument into an efficient algorithm is to use approximate sparsest cuts instead. This gives polynomial time algorithms at the cost of leaving more edges across the clusters. If we fix the conductance requirement ϕ of the expander decomposition, i.e. $\min_i \Phi_{G[V_i]} \geq \phi$, the quality of an expander decomposition algorithm is characterized by two measures: $\sum_i \delta(V_i)$, which we refer to as the *error*, and the running time. We want a fast algorithm with error almost as small as in Observation 1.1.

1.1 Previous Work

There are various efficient algorithms that can compute an approximate sparsest cut. These methods mostly fall into two categories. One type of the methods are spectral based, that is, they use the eigenvalue and eigenvectors of the graph Laplacian matrix or the random walk diffusion process. These methods can have quadratic error in the cut quality, which means that if the sparsest cut has conductance γ, the cut found by these methods can have conductance as large as $\Omega(\sqrt{\gamma})$. The quadratic loss is inherent to spectral methods as observed by Cheeger [Che69]. Another type of methods are based on single-commodity or multi-commodity flow techniques, and these flow-based methods can find cut with conductance that is at most a $\log^{O(1)} m$ factor worse than the optimal conductance ([OSVV08, ARV09, KRV09, She09]).

One can directly apply the approximate sparsest cut methods in the recursive decomposition approach, and get polynomial time expander decomposition algorithms of various error bounds ([KVV00]). However, as the approximate sparsest cut found can be very unbalanced, i.e. one side of the cut has much smaller volume than the other side, the recursion depth can be $\Omega(n)$, and thus this approach inherently takes $\Omega(n^2)$ time which is too slow for many applications.

The issue here is that black-box usage of approximate sparsest cut methods may return a very unbalanced cut even when a balanced sparse cut exists. Indeed, efficiently certifying there is no balanced sparse cut when the algorithm finds an unbalanced sparse cut is the main challenge in this line of work. Previous results achieving below quadratic running time all utilize nearly linear time subroutines to find approximately most balanced sparse cuts. In the case where the subroutines find a very unbalanced sparse cut, they also provide certificates that no sparse cut of much better balance exists. These certificates can usually be interpreted as the larger side of the unbalanced cut having certain well-connected properties, but none of them is strong enough to certify the larger side induces an expander. We discuss two representative prior results in more detail in the following. Henceforth, we use $\tilde{O}(\cdot)$ to hide a polylog(n) factor.

First, Spielman and Teng [ST04] show that one can find a low conductance cut $(S, \overline{S})$ with the following guarantee: either $(S, \overline{S})$ is balanced (i.e. $\min\{vol(S), vol(\overline{S})\} = \Omega(m)$), or the larger side $G[\overline{S}]$ is contained in *some* unknown expander subgraph. Their algorithm just settles with the weaker guarantee on $G[\overline{S}]$, and thus *avoids* recursing on the larger side of the unbalanced cut. This makes the recursion depth to be $O(\log n)$, and their algorithm takes $\tilde{O}(m/\text{poly}(\phi))$ time and has error $\sum_i \delta(V_i) = \tilde{O}(\sqrt{\phi}m)$. As $G[\overline{S}]$ may not induce an expander, their decomposition only guarantees the *existence* of some unknown set $W_i \supseteq V_i$ where $\Phi_{G[W_i]} \geq \phi$ for each cluster V_i in their output. They show that the weaker expansion guarantee is sufficient for the application of spectral graph sparsification [ST11], and their remarkable result has since become the building block in several breakthroughs of fast graph algorithms (e.g. [KLOS14, CKP+17]). However, there are other applications that crucially need the stronger guarantee of $\Phi_{G[V_i]} \geq \phi$ for each cluster, e.g. dynamic minimum spanning forest algorithms [NS17, Wul17, NSW17] and short-cycle decomposition [CGP+18].

The second result is by the independent work of Nanongkai and Saranurak [NS17] and Wulff-Nilsen [Wul17]. By using an approximate balanced sparse cut algorithm [KRV09, Pen16] in a black-box manner, they get a low conductance cut $(S, \overline{S})$ such that, if the cut is not balanced, then any

subset T of the larger side $\overline{S}$ must have high conductance in $\overline{S}$ if $vol(T) \geq k$ for some k.[1] Note that if $k = 1$, then $G[\overline{S}]$ would have been an expander. They show how to iteratively reduce k to 1 by recursing on the larger side $m^{o(1)}$ many times until they obtain an expander subgraph. This approach gives real expander decomposition (i.e. $\forall i : \Phi_{G[V_i]} \geq \phi$), takes total running time $O(m^{1+O(\log\log n/\sqrt{\log n})})$, and the final decomposition has error $\sum_i \delta(V_i) = O(\phi m^{1+O(\log\log n/\sqrt{\log n})})$.

1.2 Our contribution.

In this paper, we simultaneously improve upon both previous works. That is, similar to [ST04], we do not recurse on the larger side of the unbalanced cut, while at the same time every output cluster induces an expander like in [NS17, Wul17]. Apart from expander decomposition, our main technique also lead to better expander pruning for dynamic graphs. We discuss them separately.

Expander Decomposition Our result is the first nearly linear time expander decomposition algorithm in the regime of ϕ being at least $1/\log^{O(1)} m$, which is the case for most applications of expander decomposition [ACK+16, JS18, CGP+18, ST04, CKP+17, Tre05, NSW17].

Theorem 1.2 (Expander Decomposition). *Given a graph $G = (V, E)$ of m edges and a parameter ϕ, there is a randomized algorithm that with high probability finds a partitioning of V into $V_1, \ldots, V_k$ such that $\forall i : \Phi_{G[V_i]} \geq \phi$ and $\sum_i \delta(V_i) = O(\phi m \log^3 m)$. In fact, the algorithm has a stronger guarantee that $\forall i : \Phi_{G\{V_i\}} \geq \phi$. The running time of the algorithm is $O(m \log^4 m/\phi)$.*

Note our result also has the right dependence (up to polylog$n$ factor) on ϕ in the error bound according to Observation 1.1. The construction by Spielman and Teng [ST04], even after using the state-of-the-art spectral-based algorithms for finding sparse cuts [ACL06, AP09, OV11, GT12, OSV12], can only guarantee $\sum_i \delta(V_i) = \tilde{O}(\sqrt{\phi} m)$ due to the intrinsic Cheeger barrier of spectral methods. Beyond the theoretical improvements, we note that comparing to prior work, our result only relies on techniques that are fairly basic and simple, and thus is very likely to have practical significance.

Our result extends to weighted graphs in a fairly straightforward way as in Theorem 4.1, see Section 4.1 for details. For simplicity, we focus on the unweighted case in our presentation.

Expander Pruning Although the utility of expander decomposition is well-known for static problems, it was until recently that this graph decomposition has been utilized for dynamic problems. Nanongkai, Saranurak, and Wulff-Nilsen [NSW17] significantly improved the 20-year-old $O(\sqrt{n})$ worst-case update time [Fre85, EGIN97] of dynamic minimum spanning forest to only $n^{o(1)}$ time by using the expander decomposition. However, as the decomposition itself is a static object, they need a key tool which make this possible, called *expander pruning*. Expander pruning is an algorithm for maintaining an expander under edge deletions as follows:

Theorem 1.3 (Expander Pruning). *Let $G = (V, E)$ be a ϕ expander with m edges. There is a deterministic algorithm with access to adjacency lists of G such that, given an online sequence of $k \leq \phi m/10$ edge deletions in G, can maintain a* pruned set *$P \subseteq V$ such that the following property holds. Let G_i and P_i be the graph G and the set P after the i-th deletion. We have, for all i,*

1. *$P_0 = \emptyset$ and $P_i \subseteq P_{i+1}$,*
2. *$vol(P_i) \leq 4i/\phi$ and $|E(P_i, V - P_i)| \leq 2i$, and*
3. *$G_i\{V - P_i\}$ is a $\phi/6$ expander.*

[1] More formally, we mean $|E(T, \overline{S} - T)| \geq \phi \mathbf{vol}_{G[\overline{S}]}(T)$ for all $T \subset S$ where $k \leq \mathbf{vol}_{G[\overline{S}]}(T) \leq \mathbf{vol}(G[\overline{S}])/2$.

The total time for updating $P_0, \ldots, P_k$ is $O(k \log m/\phi^2)$.

This significantly improves in many ways the best known result by Nanongkai, Saranurak, and Wulff-Nilsen [NSW17], which in turns improve the previous algorithms in [NS17, Wul17]. In Theorem 5.2 of [NSW17]; 1) all the deletions must be given in one batch and the algorithm only outputs the set P for the graph after all edges in the batch are deleted, 2) their slower running time is $\tilde{O}(\frac{\Delta k^{1+\delta}}{\phi^{6+\delta}})$ where Δ is the max degree of G and $\delta \in (0,1)$, and 3) they only guarantee that $\Phi_{G[V-P]} = \Omega(\phi^{2/\delta})$ which is much lower than $\phi/6$.

Their result is obtained by calling a local-flow subroutine [OA14, HRW17] in a black-box way for many rounds without reusing flow information from previous rounds. On the other hand, by using the *trimming* technique in Section 3 for constructing the expander decomposition in Theorem 1.2 which can "reuse" the local flow across many rounds, Theorem 1.3 is obtained almost immediately.

Theorem 1.3 is one of very few dynamic algorithms whose amortized update time of $O(\log m/\phi^2)$ is guaranteed over a short sequence of updates (i.e. $k \leq \phi m/10$). Previous amortized update time in the literature only guarantees the total update time is $O(mT)$ for some T. This gives $O(T)$ amortized only if the sequence is of length $\Omega(m)$ (see e.g. [ES81, HK99, HdLT01]).

Again the result extends to weighted graphs in a fairly straightforward way. For long sequence of updates, we can show the first expander pruning on weighted graphs with small amortized update time (Theorem 4.2). We expect this would enable future dynamic algorithms to exploit the expander decomposition on weighted graphs. See Section 4.1 for details.

2 Overview

In this section we discuss the high-level ideas of our algorithm, and show how these lead to our main result Theorem 1.2. Our algorithm (Algorithm 1) use the recursive decomposition framework, where we try to find a sparse cut that is as balanced as possible, and recurse on both sides if the cut is balanced up to some polylog$n$ factor. However, if the sparse cut $(A, \overline{A})$ we find is very unbalanced, the key observation is that as long as we have certain weak expansion guarantee on the larger side $G[A]$ (Definition 3.1), we can efficiently find another sparse cut $(A', \overline{A'})$ such that we can certify the larger side $G[A']$ is an induced expander. Our key technical contribution is a fast and simple method to perform some local fixing in $G[A]$ around the cut $(A, \overline{A})$ to obtain such A'. As a consequence, we only need to recurse on the smaller side $G[\overline{A'}]$ in this case, and the recursion has at most $O(\text{polylog}n)$ depth, while all clusters we find are induced expanders. We refer to this method of finding A' as the *trimming* step, and formalize its performance as follows.

Theorem 2.1 (Trimming)**.** *Given graph $G = (V, E)$ and $A \subset V, \overline{A} = V \setminus A$ such that A is a nearly ϕ expander in G, $|E(A, \overline{A})| \leq \phi \mathbf{vol}(A)/10$, the trimming step finds $A' \subseteq A$ in time $O\left(\frac{|E(A,\overline{A})| \log m}{\phi^2}\right)$ such that $\Phi_{G\{A'\}} \geq \phi/6$. Moreover, $\mathbf{vol}(A') \geq \mathbf{vol}(A) - 4|E(A, \overline{A})|/\phi$, and $|E(A', \overline{A'})| \leq 2|E(A, \overline{A})|$.*

The notion of a nearly ϕ expander, which is formally defined in Definition 3.1, is the weak expansion guarantee we require for the larger side $G[A]$ of the unbalanced cut. We can usually get this guarantee from a approximate balanced sparse cut subroutine when it returns a very unbalanced cut. For example, the guarantee in the result of Spielman and Teng [ST13] that the larger side of the unbalanced cut is inside some unknown larger expander immediately implies the larger side is a nearly expander. Other methods for balanced sparse cut (e.g. [OV11, OSV12]) also provide certificates in the case of a very unbalanced cut that typically can imply the nearly expander guarantee on the larger side fairly straightforwardly. The upper-bound on $|E(A, \overline{A})|$ as a condition in Theorem 2.1 holds since the cut $(A, \overline{A})$ has low conductance and $\mathbf{vol}(\overline{A})$ is much smaller than $\mathbf{vol}(A)$ (i.e. unbalanced).

Our trimming step relies on recent development on local flow algorithms [OA14, HRW17], and adapts (in a white-box manner) the particular push-relabel based local flow algorithm from [HRW17].

Algorithm 1 Expander Decomposition

Decomp(G, ϕ)
. **Call** Cut-Matching(G,ϕ)
. **If** we certify $\Phi_G \geq \phi$, **Return** G
. **Else if** we find relatively balanced cut (A, R)
. . **Return** Decomp$(G\{A\}, \phi) \bigcup$ Decomp$(G\{R\}, \phi)$
. **Else** (i.e., we find very unbalanced cut (A, R))
. . A'=Trimming(G, A, ϕ)
. . **Return** $A' \bigcup$ Decomp$(G\{V \setminus A'\}, \phi)$.

In our trimming step, we will compute an approximate max flow in rounds, and there are possibly many rounds. The main challenge is that both the graph and the flow demands evolve across the rounds, so the running time can be very slow if we apply previous local flow methods (or nearly linear time approximate max flow) as black-box in each round, as the number of rounds can be large. Instead, we adapt our flow subroutine in a way to make it *dynamic*, so that if the graph and flow demands only change a little bit across two rounds, we can quickly update the flow solution from the previous round to get a new flow solution instead of computing from scratch. This allows us to bound the total running time as long as we can bound the total amount of change on the flow problems across all rounds rather than the total number of rounds. This is also the key insight of how our techniques give improved algorithms on dynamic graphs.

Given the trimming step, one can pair it with various balanced sparse cut methods to develop expander decomposition algorithms. To keep our discussion concrete and complete, we focus on a specific method which is a fairly standard adaptation of the *cut-matching framework* by Khandekar, Rao and Vazirani [KRV09]. The choice of this particular method is due to its simplicity and robustness. Moreover, the basic flow subroutine we use in the trimming step can be easily integrated to the cut-matching framework, so we can avoid black-box usage of a hammer such as approximate max flow, which has a horrendous polylog$n$ factor in the running time, and is also not easy to implement or even capture with simpler heuristics. We refer to our method as the *cut-matching* step, with the following guarantee.

Theorem 2.2 (Cut-Matching)**.** *Given a graph $G = (V, E)$ of m edges and a parameter ϕ, the cut-matching step takes $O((m \log m)/\phi)$ time and must end with one of the three cases:*

1. *We certify G has conductance $\Phi_G \geq \phi$.*
2. *We find a cut $(A, \overline{A})$ in G of conductance $\Phi_G(\overline{A}) = O(\phi \log^2 m)$, and $\mathbf{vol}(A), \mathbf{vol}(\overline{A})$ are both $\Omega(m/\log^2 m)$, i.e., we find a relatively balanced low conductance cut.*
3. *We find a cut $(A, \overline{A})$ with $\Phi_G(\overline{A}) \leq c_0 \phi \log^2 m$ for some constant c_0, and $\mathbf{vol}(\overline{A}) \leq m/(10c_0 \log^2 m)$, and A is a nearly ϕ expander (See Definition 3.1).*

As our cut-matching step is a fairly straightforward adaptation of the work of Khandekar et al. [KRV09], and is similar to how Räcke, Shah and Täubig [RST14] adjust the cut-matching framework in the context of oblivious routing, we defer further description and analysis of it to Appendix B. We note that the setting we apply our cut-matching step is more regularized than the one in [RST14], so our adaptation and analysis are considerably simpler and more basic comparing to [RST14].

Given the cut-matching step and the trimming step, we can combine the guarantee from case (3) of the cut-matching step (Theorem 2.2 with the trimming step (Theorem 2.1) to write out explicitly the quality of the cut from the trimming step. We have $\mathbf{vol}(\overline{A}) \leq m/(10c_0 \log^2 m)$ and $\Phi_G(\overline{A}) \leq c_0 \phi \log^2 m$ from case(3) of the cut-matching step. So $|E(A, \overline{A})| \leq \phi m/10$. Thus, the

trimming step in $O((m \log m)/\phi)$ time gives a $\phi/6$ expander $G\{A'\}$ where $\mathbf{vol}(A') \geq 3m/2$ (note the total volume is $2m$). Theorem 2.1 also indicates that the conductance of $(A', V \setminus A')$ is at most twice the conductance of $(A, V \setminus A)$, so $(A', V \setminus A')$ has conductance $O(\phi \log^2 m)$. Now we can give a quick proof of our main result.

Proof of Theorem 1.2. Since our recursive algorithm only stops working on a component when it certifies the induced subgraph has conductance at least $\phi/6$, the leaves of our recursion tree give an expander decomposition $V_1, \ldots, V_k$ of V, and $\Phi_{G\{V_i\}} \geq \phi/6$ for all $i \in [1, k]$. Note that whenever we cut a component and recurse, we always add self-loops so the degree of a node remains the same as its degree in the original graph, so we get the stronger expansion guarantee with respect to the volume in the original graph.

We now bound the running time. After carrying out the cut-matching step on a component, if we get case (1) in Theorem 2.2, we are done with the component. In case (2), we get a relatively balanced sparse cut, and we recurse on both sides of the cut. In case (3), we use the trimming step to get a sparse cut such that the larger side of the cut is a $\phi/6$ expander, and we only recurse on the smaller side of the cut. In any case, we get the volume of the largest component drops by a factor of at least $1 - \Omega(1/\log^2 m)$ across each level of the recursion, so the recursion goes up to $O(\log^3 m)$ levels. As the components on one level of the recursion are all disjoint, the total running time on all the components of one level of the recursion is $O((m \log m)/\phi)$, so the total running time is $O\left((m \log^4 m)/\phi\right)$.

To bound the number of edges between expander clusters, observe that in both case (2) and case (3), we always cut a component along a cut of conductance $O(\phi \log^2 m)$. Thus, we can charge the edges on the cut to the edges in the smaller side of the cut, so each edge is charged $O(\phi \log^2 m)$. An edge can be on the smaller side of a cut at most $\log m$ times, so we can charge each edge at most $O(\phi \log^3 m)$ to pay for all the edges we leave between the final clusters. This bound the total number of edges between the expanders to be at most $O(m\phi \log^3 m)$.

In the rest of this extended abstract, we discuss the trimming step in Section 3 with some details of the flow subroutine deferred to Appendix A. We briefly discuss the (theoretical) applications of our result and open problems in Section 4. For completeness, we discuss the cut-matching step in Appendix B. We will keep our discussion at a high level, and leave most of the technical details to the full version.

3 The Trimming Step

In this section, we describe the trimming step and prove Theorem 2.1, which is the key technical contribution of this paper. We start with defining a nearly expander formally and introducing the flow terminology we use in our subroutine.

3.1 Preliminaries

Definition 3.1 (Nearly Expander)**.** *Given $G = (V, E)$ and a set of nodes $A \subset V$, we say A is a* nearly ϕ expander *in G if*

$$\forall S \subseteq A, \mathbf{vol}(S) \leq \mathbf{vol}(A)/2 : |E(S, V \setminus S)| \geq \phi \mathbf{vol}(S)$$

Note if the left hand side of the inequality is $|E(S, A \setminus S)|$, $G\{A\}$ would be a ϕ expander.

The trimming step aims to find a $A' \subseteq A$ that $G\{A'\}$ is a $\phi/6$ expander, i.e.,

$$\forall S \subseteq A', \mathbf{vol}(S) \leq \mathbf{vol}(A')/2 : |E(S, A' \setminus S)| \geq \phi \mathbf{vol}(S)/6$$

Recall when we consider induced subgraphs, we always add self-loops to nodes so that the degree of a node is the same as its degree in the original graph, so there is no ambiguity on the notation $\mathbf{vol}(S)$.

A *flow problem* Π on a graph $G = (V, E)$ is specified by a source function $\Delta : V \to \mathbb{R}_{\geq 0}$, a sink function $T : V \to \mathbb{R}_{\geq 0}$, and edge capacities $c : E \to \mathbb{R}_{\geq 0}$. We use *mass* to refer to the substance being routed. For a node v, $\Delta(v)$ specifies the amount of mass initially placed on v, and $T(v)$ specifies the capacity of v as a sink. For an edge e, $c(e)$ bounds how much mass can be routed along the edge.

A *routing* (or *flow*) $f : V \times V \to \mathbb{R}$ satisfies $f(u, v) = -f(v, u)$ and $f(u, v) = 0$ for $\{u, v\} \notin E$. $f(u, v) > 0$ means that mass is routed in the direction from u to v, and vice versa. If $f(u, v) = c(e)$, then we say (u, v) is *saturated* (in the direction from u to v). Given Δ, we also treat f as a function on vertices, where $f(v) = \Delta(v) + \sum_u f(u, v)$ is the amount of mass ending at v after the routing f. If $f(v) \geq T(v)$, then we say v's sink is *saturated*.

f is a *feasible routing/flow* for Π if $|f(u, v)| \leq c(u, v)$ for each edge $e = \{u, v\}$ (i.e. obey edge capacities), $\sum_u f(v, u) \leq \Delta(v)$ for each v (i.e. the net amount of mass routed away from a node can be at most the amount of its initial mass), and $f(v) \leq T(v)$ for each v. These notations are more natural in the discussion of local flow methods.

3.2 Slow Trimming

Now we discuss the intuition of why the trimming step is possible, and give a clean way to perform it using exact max flow computation which takes super-linear time (i.e. slow trimming).

The key observation is very simple, which basically says that if A is a nearly expander in G, but $G\{A\}$ is not an induced expander, then any bottleneck in $G\{A\}$ must be close to the places where A is cut off from G. More formally, consider the following flow problem in $G\{A\}$. We let each edge in $E(A, V \setminus A)$ be a source of $2/\phi$ units of mass[2], each node v be a sink of capacity equal to its degree $\deg(v)$, and edges all have capacity $2/\phi$. The following observation motivate our algorithm for the trimming step:

Proposition 3.2. *Suppose that $A \subset V$ is a nearly ϕ expander in G, but $G\{A\}$ is not a $\phi/6$ expander. Then, the flow problem constructed as above doesn't have a feasible routing.*

Proof. Given the assumptions, there must be a $S \subset A, \mathbf{vol}(S) \leq \mathbf{vol}(A)/2$ such that

$$\frac{|E(S, A - S)|}{\mathbf{vol}(S)} \leq \phi/6,$$

but

$$\frac{|E(S, V - S)|}{\mathbf{vol}(S)} = \frac{|E(S, A - S)| + |E(S, V - A)|}{\mathbf{vol}(S)} \geq \phi.$$

Let $a = |E(S, V - A)|, b = |E(S, A - S)|$, the above two inequalities give $a \geq 5b$ and $\mathbf{vol}(S) \leq \frac{6a}{5\phi}$. That is, any sparse cut S in $G\{A\}$ must be local to where A is cut off from the rest of the graph.

Each of the $a = |E(S, V - A)|$ edges is a source of $2/\phi$ units of mass, so the total amount of source mass started in S is $2a/\phi$. However, the total sink capacity of nodes in S is $\mathbf{vol}(S) \leq 6a/(5\phi)$, and the amount of mass that can be routed out of S to $A - S$ along the $b = |E(S, A - S)|$ edges is at most $2b/\phi \leq 2a/(5\phi)$ due to edge capacity. There is too much initial mass in S that we cannot route all the mass to sinks in S or out of S under edge and sink capacities as $2a/\phi > 6a/(5\phi) + 2a/(5\phi)$, and thus there can be no feasible routing of the flow problem we construct. □

[2] We slightly abuse the notation to place initial mass on an edge that no longer exists in $G\{A\}$. As a cut edge $\{u, v\} \in E(A, V \setminus A)$ with $v \in A$ corresponds to a self-loop attached to v in $G\{A\}$, technically v is where we actually place the initial mass. Note if x edges in $E(A, V \setminus A)$ are incident to v, the amount of source mass on v will be $2x/\phi$ units.

Algorithm 2 Trimming

Trimming(G,A,ϕ)
. Set $A' = A$
. **While** $G\{A'\}$ is not a certified $\phi/6$ expander
. . Construct the flow problem in $G\{A'\}$ where
. . . Each edge in $E(A', V - A')$ is a source of $2/\phi$ units of mass,
. . . Each node $v \in A'$ is a sink of capacity $\deg(v)$,
. . . Each edge in $G\{A'\}$ has capacity $2/\phi$.
. . Use a flow algorithm to find a feasible routing.
. . **If** a feasible routing is found
. . . $G\{A'\}$ is a certified $\phi/6$ expander.
. . **Else** (i.e. a cut is found in $G\{A'\}$ with S being the small side)
. . . $A' = A' - S$

Flow problems where each node is a sink with capacity proportional to its degree have been used previously in the literature (e.g. for finding densest subgraphs [Gol84] and for improving cut quality [LR04, AL08, OA14, VGM16, WFH+17]). Note that Proposition 3.2 holds if we use sink capacity $T(v) = c\deg(v)$ for any $c \leq 1$. The scalar c dictates how local the flow computation needs to be, as long as c is not too small so there is no feasible routing due to the trivial reason that the total sink capacity over the entire graph is not enough for the source mass. In our setting, the nearly expander guarantee allows us to use a fairly large $c = 1$ and still recognize any sparse cut in $G\{A\}$ as long as there exists one. Thus, our flow computation can be very efficient, since it only needs to explore region very local to where we put initial mass. This local feature of our flow problem is why the running time of our trimming step is proportional to $|E(A, V \setminus A)|$ instead of $|E(G\{A\})|$, and makes it useful in the dynamic setting. We note this idea has be exploited before to design local algorithms, for example, the local improve algorithm by Orecchia and Zhu [OA14] comparing to the global improve algorithm by Andersen and Lang [AL08]. Intuition similar to our key observation Proposition 3.2 is also exploited by [NS17, NSW17] for bounded degree graphs.

Given Proposition 3.2, we can take a generic approach for our trimming step as in Algorithm 2. We proceed in rounds, where we start with $A' = A$ in the first round, construct our flow problem in $G\{A'\}$, and use some flow algorithm to route the initial mass to sinks. If a feasible routing is found, as any subset A' of A is inherently a nearly ϕ expander in G, Proposition 3.2 certifies that $G\{A'\}$ is a $\phi/6$ expander. If the flow algorithm doesn't find a feasible routing, we will get a cut S in $G\{A'\}$. Note in general we won't use an exact max flow algorithm for efficiency purpose, this case can happen even when a feasible routing exists, and the returned cut S can be an approximate min cut instead of the min cut. In this case we trim S (i.e. remove nodes in S and their incident edges) from A', and proceed to the next round. We do this iteratively until in some round our flow algorithm finds a feasible routing for the flow problem defined on $G\{A'\}$ to certify the $G\{A'\}$ in that round is a $\phi/6$ expander.

We first look at the extreme case where we use a exact max flow algorithm to compute the flow, and show that the trimming step must finish in at most 2 rounds. In this case we will temporarily switch to an $s - t$ max-flow formulation, and translate the standard max-flow min-cut property to our flow language.

Our flow problem in max-flow notation: The flow problem in our discussion is equivalent to a $s-t$ max-flow problem in the augmented network of $G\{A\}$ where we add a super-source s, a super-sink t to $G\{A\}$, add a directed edge of capacity $2/\phi$ from s to each source in $E(A, V \setminus A)$, add a directed edge of capacity $\deg(v)$ from each $v \in A$ to t, and each original (undirected) edge in $G\{A\}$ has capacity $2/\phi$.

Routing x units of mass out of a source e is equivalent to sending x flow along the edge from s to e, and routing x units of mass to the sink of a node v is equivalent to sending x flow along the edge from v to t. Routing all the mass from sources to sinks (i.e. a feasible routing in our notation) is equivalent to the $s-t$ max-flow problem having max-flow value $\frac{2|E(A,V-A)|}{\phi}$.

If we are not done after the first round of trimming, we will have a max-flow f, and a corresponding min-cut S in the augmented network of $G\{A\}$, where S is the side containing the super-source s. The important fact about exact max-flow and min-cut is that f will saturate all edges going out of S. This implies the following:

Claim 3.3. The standard max-flow min-cut properties translate f and S to a routing and cut in our notation with the following properties.

1. We can route $2/\phi$ units of mass from every source in $A \setminus S$ to sinks, because in the augmented network the edge from s to any such source crosses the min-cut, so f sends $2/\phi$ flow along the edge. Moreover, as f only sends flow out of S, the mass from any source in $A \setminus S$ must be routed to sinks in $A \setminus S$ using only edges in $G\{A \setminus S\}$.

2. We can route $2/\phi$ units of mass from every edge in $E(S, A \setminus S)$ to sinks in $A \setminus S$ using only edges in $G\{A \setminus S\}$. Again this is due to f saturating every edge across the cut from S to $A \setminus S$.

3. Any node $v \in A$ that falls in S must have its sink saturated, i.e. it receives $\deg(v)$ units of mass. This is because the edge (v, t) in the augmented network crosses the min-cut.

In the second round, we have $A' = A \setminus S$. Consider the flow problem we construct on $G\{A'\}$ in the second round: the sources are the edges in $E(A', V \setminus A')$, which are of one of the two types

1. Sources in $E(A', V \setminus A') \bigcap E(A, V \setminus A)$: These are also sources in the first round that falls in $A \setminus S$. By Claim 3.3(1), we can route $2/\phi$ units of mass from each of of these sources to sinks in $G\{A'\}$.

2. Sources in $E(A', V \setminus A') - E(A, V \setminus A)$: These new sources are exactly the cut edges in $E(S, A \setminus S)$. By Claim 3.3(2), we can route $2/\phi$ units of mass from each of of these sources to sinks in $G\{A'\}$.

Thus, the flow routing from the first round already gives us a feasible routing in the second round, and we can certify $G\{A'\}$ is a $\phi/6$ expander. Moreover, by Claim 3.3(3) every node v in $S = A \setminus A'$ receives $\deg(v)$ units of mass in the routing of first round, so the total volume of the nodes we remove from A is bounded by the total amount of mass $2|E(A, V - A)|/\phi$, which gives $\mathbf{vol}(A') \geq \mathbf{vol}(A) - 2|E(A, V - A)|/\phi$. Moreover, as the routing in the first round routes $2/\phi$ units of mass out of every edge in $E(A', V \setminus A')$, and these mass are distinct since f only routes mass from S to $A \setminus S$, so $2|E(A', V \setminus A')|/\phi$ is also bounded by the total amount of mass $2|E(A, V - A)|/\phi$, which gives $|E(A', V - A')| \leq |E(A, V - A)|$.

The above discussion already gives (even a stronger version of) everything we want in Theorem 2.1 except the running time. Using exact max flow in the trimming step is too slow for us because currently all the algorithms that solves max-flow exactly takes $m^{1+\Omega(1)}$ time ([LS14, Mad16]). Nonetheless, this gives some intuition why trimming in general can terminate and give guarantee as in Theorem 2.1 even if we use approximate flow computation.

3.3 Efficient Trimming

As we cannot afford to use exact max flow for efficiency purpose, in general we can only expect to find a pair of flow and cut that are approximately tight, that is, the flow saturates most of the outgoing capacity of the cut. As a result, there is no guarantee that if we remove a cut, the flow problem in the next round will have a feasible routing as in the exact max flow case. The main challenge is that the trimming step can take many rounds to converge, and although approximate max flow only takes nearly linear time to compute, it will be too slow for us to solve the flow problem in each round from scratch. This is the main reason it is not sufficient to directly apply efficient approximate max flow methods such as [KLOS14, She13, OA14, HRW17].

Consider the sequence of flow problems in our trimming step, we can view it as a dynamic flow problem, where in each round we update the graph by trimming the cut found in the last round, and add new source mass to the graph. Thus, we need a flow subroutine to handle these flow problems dynamically instead of starting from scratch in each round. Note the changes to the flow problems are quite regularized, where the graph strictly shrinks across rounds, any source from one round either remains a source in the next round or is removed from the graph, and new sources are added exactly at the new cut edges. In the extreme case of exact max flow min cut, the argument is simple because of the strong optimality condition where the max flow f saturates every edge from S to $A \setminus S$. This allows us to reuse the flow in a trivial manner by constraining the routing f to the subgraph $G\{A \setminus S\}$. If we only compute an approximately tight pair of flow and cut (instead of exact max-flow and min-cut), we get weaker optimality conditions such as the routing saturates most of the outgoing capacity of the cut (instead of all of the outgoing capacity). When we proceed from round i to round $i+1$ with the cut removed, the routing still tells us how to route *most* of the source mass in round $i+1$. This strongly suggests we should be able to reuse the flow routing across rounds, and only pay update time proportional to how much the flow problems have changed.

3.3.1 Reusing Flow Information

To really make an approximate flow method work in our dynamic setting, we need much more detailed knowledge of the pair of flow and cut we compute in addition to that they are approximately tight. Thus, we adapt the local flow method *Unit-Flow* by Henzinger, Rao, and Wang [HRW17], which is based on the Push-Relabel framework [GT88].

Why push-relabel? The reason behind this choice is that the flow and cut computed by *Unit-Flow* have certain nice invariants that make reusing flow routing across rounds very simple in terms of both operation and analysis. We need amortization in our analysis; the worst-case cost of one round maybe very high but they are fast amortized over all rounds. The potential function based analysis of *Unit-Flow* is very natural to adapt for the amortized analysis.

It is conceivable that one can adapt blocking-flow based methods such as [OA14] to be dynamic, but the two-level structure of Dinic's algorithm (i.e. multiple rounds of blocking flow computations) makes it more difficult to carry out the adaptation and running time analysis. In particular, blocking-flow based algorithms do not have the flexibility as push-relabel to extend naturally to amortized running time analysis.

Note that although we start with an undirected graph, if we want to reuse flow routing across rounds, it is natural to work with the residual network which is directed. Thus, it is not clear how to adapt approximate max flow algorithms designed for undirected graphs[3] ([KLOS14, She13]) to our dynamic setting.

[3] whereas push-relabel based methods naturally work with directed graphs.

3.3.2 Dynamic Adaptation of *Unit-Flow*

We give a quick overview of the *Unit-Flow* algorithm, which is a particular adaptation of the generic Push-Relabel algorithm. A key concept is a *pre-flow*: Given a flow problem $\Pi = (\Delta, T, c)$, a pre-flow f is a feasible routing for Π except the condition $\forall v : f(v) \leq T(v)$. As pre-flow may not obey sink capacity on nodes, we call the *absorbed* mass on a node v as $\text{ab}(v) = \min(f(v), T(v))$. We have $\text{ab}(v) = T(v)$ iff v's sink is saturated. The *excess* on v is $\text{ex}(v) = f(v) - \text{ab}(v)$. Observe that when there is no excess, $\forall v : \text{ex}(v) = 0$, f is a feasible flow for Π.

As in the generic Push-Relabel framework, each node v has a non-negative integer label $l(v)$, which starts at 0 and only increases throughout the algorithm, and we say a node v is at level i if $l(v) = i$. We also maintain the standard residual network , where each undirected edge $\{u, v\}$ corresponds to two directed arcs $(u, v), (v, u)$, and the residual capacity of an arc (u, v) is $r_f(u, v) = c(\{u, v\}) - f(u, v)$.

In the generic Push-Relabel algorithm, an arc (u, v) is *eligible* if $r_f(u, v) > 0$ and $l(u) = l(v) + 1$. The generic algorithm picks any node u with excess mass, and tries to push the excess mass away from u along eligible arcs. If there is no eligible arcs adjacent to u to push the excess, the algorithm raises $l(u)$ by 1. The algorithm keeps doing this until there is no excess on any node.

Unit-Flow uses a parameter h to upper-bound the labels on nodes, and simply leaves the excess mass on a node once it is raised to level h. This makes *Unit-Flow* very efficient, but it may only find a pre-flow even when a feasible flow exists. However, picking an appropriate value of h allows us to find a pair of pre-flow and cut that are sufficient for our purpose. We note similar intuition is exploited previously in [OA14] on Dinic's algorithm.

Let $|\Delta(\cdot)| = \sum_v \Delta(v)$ denote the total amount of initial mass. We omit the subscript when it is clear. From now, we always consider flow problems where $\forall v : T(v) = \deg(v)$ and $\forall e : c(e) = 2/\phi$ so we leave them implicit.

Given a graph $G = (V, E)$, a parameter h, and a source function Δ as inputs, in the execution of *Unit-Flow*, it maintains a pre-flow f and labels on nodes $l : V \to \{0, \ldots, h\}$ with the following invariant:

1. If $l(u) > l(v) + 1$ where (u, v) is an edge, then (u, v) is saturated in the direction from u to v, i.e. $f(u, v) = 2/\phi$.

2. If $l(u) \geq 1$, then u's sink is saturated, i.e. $f(u) = \Delta(u) + \sum_v f(v, u) \geq \deg(u)$.

We say that a tuple (Δ, f, l) is a *G-valid* state if it satisfies the invariant above. This notion captures what a intermediate state in the execution of *Unit-Flow*must obey (in addition to the conditions of a pre-flow). Furthermore, we say a tuple (Δ, f, l) is a *G-valid* solution if it satisfies the additional invariant

3. If $l(u) < h$, then there is no excess mass at u, i.e. $f(u) = \Delta(u) + \sum_v f(v, u) \leq \deg(u)$.

A G-valid solution is what *Unit-Flow* at termination must obey, since we only stop pushing excess when a node is raised to level h. Note by the last two invariants, we know $f(u) = \deg(u)$ if $1 < l(u) < h$ in a G-valid solution.

The dynamic version. Normally, when we initialize *Unit-Flow*, the pre-flow f is set as empty, i.e. $f(u, v) = 0, \forall u, v$, and as well as the labels $l(u) = 0, \forall u$. But *Unit-Flow* can in fact "'warm start"' on G given any G-valid (Δ, f, l) since it is a valid intermediate state of the algorithm, and return a G-valid solution. We can use the pre-flow and node labels to find a cut. The lemma below makes this precise. As this can be derived directly from the analysis in [HRW17], we give the proof for completeness in Appendix A.

Lemma 3.4. *Given a graph $G = (V, E)$ where $m = |E|$, a parameter h and a G-valid state (Δ, f, l) where $|\Delta| \leq 3m/2$,* Unit-Flow *outputs a G-valid solution (Δ, f', l') and a set $S \subseteq V$ where we must have one of the following two cases:*

1. *$S = \emptyset$ and f' is a feasible flow for Δ, or*

2. *$S \neq \emptyset$ is a level cut, i.e., $S = \{u \in V \mid l'(u) \geq k\}$ for some $k \geq 1$. Moreover, the number of edges (u, v) such that $u \in S, v \in V \setminus S$ and $-2/\phi \leq f'(u, v) < 2/\phi$ is at most $\frac{5\mathbf{vol}(S) \ln 2m}{h}$. Note these edges include all the edges across the cut except those sending exactly $2/\phi$ units of mass* from S into $V \setminus S$.

When we apply Lemma 3.4 to the flow problem in any round of the trimming step, if we get case (1), the trimming step can terminate since a feasible flow certifies the graph is a $\phi/6$ expander. If we get case 2, the bound on the number of unsaturated edges across the cut $(S, V \setminus S)$ limit the amount of computation in the next round when we remove S, since for the new sources corresponding to the saturated cut edges, f' already routes $2/\phi$ units of mass from them to the part of the graph not removed.

Given (G, A, ϕ) as inputs from Theorem 2.1, we implement Algorithm 2 by using Lemma 3.4 for finding a flow (and a cut) in each round. It only remains to describe the parameters we feed to Lemma 3.4. Instead of using the variable A' as in Algorithm 2, we use $A_1, A_2, \ldots$ where A_t denotes A' in round t, and use R to denote $\overline{A}$.

The parameter h is fixed to be $h = \frac{40 \ln 2m}{\phi}$ for every round. Initially, we set $A_1 = A$. Set $f_1(u, v) = 0$ and $l_1(u) = 0$ for all $u, v \in A$. Set each edge from $E(A, R)$ as a source of $2/\phi$ units. Formally, for each $u \in A$, $\Delta_1(u) = 2/\phi \times |\{e \in E(A, R) \mid u \in e\}|$. Recall that each node u is a sink of capacity $\deg(u)$, and each edge has capacity $2/\phi$. At round t, *Unit-Flow* is given as inputs $G\{A_t\}$ and (Δ_t, f_t, l_t) which is a $G\{A_t\}$-valid state, and then outputs a $G\{A_t\}$-valid solution (Δ_t, f'_t, l'_t) and a set S_t. These outputs from round t is used to defined the inputs for the next round as follows.

Operations between round t and $t + 1$

1. Let (Δ_t, f'_t, l'_t) be the $G\{A_t\}$-valid solution and S_t be the level cut outputted by *Unit-Flow* at round t.

2. Set $A_{t+1} \leftarrow A_t - S_t$.

3. Let f_{t+1} and l_{t+1} be obtained from f'_t and l'_t by restricting their domain from $A_t \times A_t$ to $A_{t+1} \times A_{t+1}$ and from A_t to A_{t+1}, respectively.

4. For each edge $e = (v, u) \in E(S_t, A_{t+1})$ where $u \in A_{t+1}$, set e to be a source of $2/\phi$ units. More formally, for each $u \in A_{t+1}$,

$$\Delta_{t+1}(u) \leftarrow \Delta_t(u) + \frac{2}{\phi} \times |\{e \in E(S_t, A_{t+1}) \mid u \in e\}|.$$

5. Use $(\Delta_{t+1}, f_{t+1}, l_{t+1})$ as input for *Unit-Flow* at round $t + 1$.

Analysis. First, we show that the given parameters are applicable for the algorithm from Lemma 3.4.

Lemma 3.5. *For each round t, (Δ_t, f_t, l_t) is a $G\{A_t\}$-valid state.*

Proof. We prove by induction. For round 1, the tuple (Δ_1, f_1, l_1) is trivially $G\{A_1\}$-valid as $A_1 = A$. For the inductive step, by Lemma 3.4, we have that the output (Δ_t, f'_t, l'_t) is $G\{A_t\}$-valid. As f_{t+1} and l_{t+1} are just a restriction of f_t and l_t, the first condition to being $G\{A_{t+1}\}$-valid holds. Indeed, for an edge (u, v) where $u, v \in A_{t+1}$, if $l_{t+1}(u) > l_{t+1}(v) + 1$, then $l'_t(u) > l'_t(v) + 1$ and so $f_{t+1}(u, v) = f'_t(u, v) = 2/\phi$.

The second condition is a little tricky. In step 3 when we restrict f'_t to the domain $A_{t+1} \times A_{t+1}$, we actually also change the amount of mass on nodes. Since if a node $u \in A_{t+1}$ is adjacent to an edge $e \in (S, A_{t+1})$, when we go from f'_t to f_{t+1}, the mass routed along e into (or out of) u is no longer counted when we compute mass on u. However, this reduce the mass at u by at most $2/\phi$, and we add $2/\phi$ of mass as initial mass to $\Delta_{t+1}(u)$ for each cut edge, thus the amount of mass is non-decreasing for the nodes remaining in A_{t+1}

$$\begin{aligned} f_{t+1}(u) =& \Delta_{t+1}(u) + \sum_{(v,u)\in E(A_{t+1})} f_{t+1}(v,u) \\ \geq& \Delta_t(u) + \sum_{(v,u)\in E(S_t,A_{t+1})} f'_t(v,u) + \\ & \sum_{(v,u)\in E(A_{t+1})} f'_t(v,u) \\ =& \Delta_t(u) + \sum_{(v,u)\in E(A_t)} f'_t(v,u) = f'_t(u) \geq \deg(u). \end{aligned}$$

The inequality is due to the difference between $\Delta_{t+1}(u)$ and $\Delta_t(u)$ and $f'_t(v,u) \leq 2/phi$. The second to last equality is because nodes incident to u in A_t must be either in S_t or A_{t+1}. □

Consider any edge $e = (u, v)$ across the cut with $u \in S_t, v \in A_{t+1}$. In the flow problem of round $t+1$ in the trimming step, we need to route $2/\phi$ initial mass from e to sinks in A_{t+1}. What we are doing between the rounds is indeed reusing the flow routing f'_t. Note the contribution of cutting e to the change of mass on v[4] is exactly $2/\phi - f'_t(u,v)$. In particular, if f'_t already routes $2/\phi$ mass from u to v, we simply reuse the routing as if these are the $2/\phi$ units of initial mass out of e. If $0 \leq f'_t(u,v) < 2/\phi$, then f'_t only tells us how to route $f'_t(u,v)$ of the $2/\phi$ units of initial mass mass out e into A_{t+1}, so we need to add the remaining mass on v and let *Unit-Flow* to further route these mass starting from the state f_{t+1}. When $f'_t(u,v) < 0$, the pre-flow f'_t actually routes mass in the wrong direction, that is, some of the initial mass from sources remaining in A_{t+1} routes mass into S_t. However, in the flow problem of round $t+1$, we already removed S_t from our graph, the routing of these mass given by f'_t is no longer valid. Thus, we have to truncate the old routing of these mass at the cut edge e, and let *Unit-Flow* to reroute these mass from e back into A_{t+1} (instead of into S_t as in f'_t). Thus, the amount of mass we add to v is more than $2/\phi$ to reflect the mass we need to reroute in addition to the $2/\phi$ initial mass we put on e as a new source.

The crucial fact from the above discussion is that the mass at a node v only increases from the removal of a cut edge (u,v) such that $f'_t(u,v) \leq 2/\phi$, and the increment of mass at v is at most $4/\phi$ from one cut edge since $f'_t(u,v) \geq -2/\phi$. We will treat the mass we add to nodes between round t and $t+1$ as mass created between the rounds. Let $|\Delta_{total}(\cdot)|$ denote the total amount of mass we create over all rounds, i.e. the mass placed before round 1 plus the mass added between rounds. Note as we remove S_t between the rounds, the mass on nodes in S_t are removed along with S_t, so we also "'destroy"' existing mass along the way. The lemma below is the key lemma for proving both correctness and running time of the algorithm.

Lemma 3.6. $\mathbf{vol}(\cup_{t\geq 1} S_t) \leq |\Delta_{total}(\cdot)| \leq 4|E(A,R)|/\phi$.

Proof. First, $|\Delta_1(\cdot)| = 2|E(A,R)|/\phi$ is the amount of initial mass in the first round, as each cut edge in $E(A,R)$ is a source of $2/\phi$ units of mass. Next, for each round $t \geq 1$, let S_t be the level cut outputted from Lemma 3.4. We will prove that before proceeding to round $t+1$: (1) the amount of mass that we added is at most $\mathbf{vol}(S_t)/2$, but (2) the amount mass that we destroy is at least $\mathbf{vol}(S_t)$. This implies that $\mathbf{vol}(\cup_{t\geq 1} S_t)$ is upper-bounded by the total amount of destroyed mass,

[4] i.e. e's contribution to $f_{t+1}(v) - f'_t(v)$

which is clearly at most the total amount of mass $|\Delta_{total}(\cdot)|$ created over all rounds. Moreover, for every unit of mass that we add, we destroy at least two units of mass. This allows us to charge every two units of mass we create to a distinct unit of existing mass, so $|\Delta_{total}(\cdot)| \leq 2|\Delta_1(\cdot)| = 4|E(A,R)|/\phi$.

Now, we prove (1). By Lemma 3.4, the number of edges where $v \in S_t$, $u \in A_{t+1}$, and $f'_t(v,u) < 2/\phi$ is at most $\frac{\mathbf{vol}(S)\ln 2m}{h} = \phi\mathbf{vol}(S_t)/8$. For each such edge, the amount of new mass added is $2/\phi - f'_t(v,u) \leq 4/\phi$. So the total added amount is $\frac{\phi\mathbf{vol}(S_t)}{8} \times \frac{4}{\phi} = \mathbf{vol}(S_t)/2$. For (2), note S_t is a level cut, so any $v \in S_t$ has $l'_t(v) > 1$. As (Δ_t, f'_t, l'_t) is valid, $f'_t(v) \geq \deg(v)$ by the second invariant of valid solutions for each $v \in S_t$. Once, we remove S_t, all the mass on any $v \in S_t$ (either in the sinks or as excess mass) is gone forever, so we destroy at least $\mathbf{vol}(S_t)$ units of mass. □

The running time analysis from [HRW17] shows that *Unit-Flow* takes $O(|\Delta_1(\cdot)|h)$ in the first round. This analysis extends seamlessly to the dynamic version.

Lemma 3.7. *The total running time of* Unit-Flow *in our trimming step over all rounds is* $O(|\Delta_{total}(\cdot)|h) = O(|E(A,R)|\log m/\phi^2)$.

On a very high level, we can charged all operations of *Unit-Flow* to the mass we create over all rounds, and each unit of mass will be charged at most $O(h)$ in total. This is a direct consequence since *Unit-Flow* relabels any node at most h times. Once a node reaches level h in some round t, either we find a feasible flow in that round so the trimming step terminates, or the node is removed with the level cut S_t since S_t must contain all nodes in the highest level h. For readers familiar with the analysis of Push-Relabel, it comes as no surprise that our running time is proportional to h as the analysis of Push-Relabel is centered around how many times a node can be relabeled. The formal proof requires understanding the implementation of *Unit-Flow* and is deferred to Appendix A.

Proof of Theorem 2.1. Lemma 3.7 bounds the time. Let $A' = A - \cup_{t\geq 1} S_t$. By Lemma 3.6, $\mathbf{vol}(A') \geq \mathbf{vol}(A) - 4|E(A,R)|/\phi$. Suppose the algorithm finishes at round T, i.e. $A' = A_T$. We get a feasible flow on $G\{A'\}$ where every edge $E(A', V' - A)$ can send mass of $2/\phi$ units to sinks in $G\{A'\}$. But the total amount of mass is at most $4|E(A,R)|/\phi$. So $E(A', V' - A) \leq (\phi/2) \cdot 4|E(A,R)|/\phi = 2|E(A,R)|$. By Proposition 3.2, we also have $\Phi_{G\{A'\}} \geq \phi/6$.

4 Discussion

4.1 Weighted Graphs

Our results extend to weighted graphs in a straightforward manner. The cut-matching framework can be adapted to weighted graphs in a standard way. As to our dynamic flow subroutine, we need to use link-cut tree (i.e. dynamic tree) data structure for weighted graphs to make the running time proportional to the total number of edges rather than the total weight. Our dynamic flow subroutine in the trimming step works well with link-cut tree. The reason is that between rounds we remove level cuts, that is, we find some level k, and remove all nodes above the level k. For readers familiar with the application of dynamic tree in Push-Relabel algorithm, it is easy to see that to maintain the cut-link tree data structure across rounds, the removal of nodes in level cuts correspond to cut sub-trees in the data structure, which is easy to carry out. We state the results for weighted graphs, and refrain from giving a more tedious analysis.

Theorem 4.1 (Weighted Graph)**.** *There is a randomized algorithm that with high probability given a graph* $G = (V, E, w)$ *with* m *edges and total weight* $W = \sum_e w_e = m^{O(1)}$ *and a parameter* ϕ*, partitions* V *into* $V_1, \ldots, V_k$ *in time* $O(m\log^5 m/\phi)$ *such that* $\min_i \Phi_{G\{V_i\}} \geq \phi$ *and* $\sum_i \delta(V_i) = O(\phi W \log^3 m)$[5].

[5]Here, volume and cut-size are defined as the sum of edges' weights rather than number of edges.

Theorem 4.2 (Weighted Expander Pruning)**.** *Let $G = (V, E, w)$ be a ϕ expander with m edges and total weight $W = \sum_e w_e$. There is a deterministic algorithm with access to adjacency lists of G such that, given an online sequence of edge deletions in G whose total weight of such edges is at most $\phi W/10$, can maintain a* pruned set *$P \subseteq V$ such that the following property holds. Let G_i and P_i be the graph G and the set P after the i-th deletion. Let W_i be the total weight of deleted edges up to time i. We have, for all i,*

1. *$P_0 = \emptyset$ and $P_i \subseteq P_{i+1}$,*
2. *$vol(P_i) \leq 4W_i/\phi$ and $|E(P_i, V - P_i)| \leq 2W_i$, and*
3. *$G_i\{V - P_i\}$ is a $\phi/6$ expander.*

The total time for updating $P_0, \ldots, P_k$ is $O(m \log m/\phi)$.

It is impossible to make the amortized update time of Theorem 4.2 hold for short update sequences. This is because the conductance can change drastically just by deleting an edge with very high weight.

4.2 Applications

In this section, we list some applications which are implied by our new expander decomposition and expander pruning in a fairly black-box manner.

User-friendly expander decomposition. The first construction of spectral sparsification of graphs by Spielman and Teng [ST04] is based on their expander decomposition. As the clusters from their decomposition are only guaranteed to be contained inside *some* expander, this complicates the analysis of their sparsification algorithm. Theorem 1.2 gives a more "user-friendly" decomposition as each of our clusters induces an expander. By using Theorem 1.2, both the algorithm for spectral sparsification and its analysis are very simple[6].

The same complication arises in the analysis of several algorithms which use Spielman and Teng's decomposition (e.g. spectral sketches [JS18], undirected/directed Laplacian solvers [ST04, CKP+17], and approximate max flow algorithms [KLOS14]). By using Theorem 1.2, one can simplify the analysis of all these algorithms.

Balanced low-conductance cut. If a graph $G = (V, E)$ has $\Phi_G < \phi$, then we can find in $\tilde{O}(m/\phi)$ a cut $(S, \overline{S})$ with conductance $O(\phi \log^3 m)$ such that the volume of the smaller side S is as large as any cut $(T, \overline{T})$ with conductance at most ϕ up to a constant factor. This is immediate from a 2ϕ expander decomposition of G; If all clusters have volume at most $9/10$ of the total volume, then we can easily group clusters to get a cut $(S, \overline{S})$ where $\Phi(S) = \tilde{O}(\phi)$ and $\min\{\mathbf{vol}(S), \mathbf{vol}(\overline{S})\} = \Omega(|E|)$. Otherwise, there is one giant cluster A. Let $R = V - A$, and it is easy to see from the proof of Theorem 1.2 that $\Phi(R) = \tilde{O}(\phi)$. As $G\{A\}$ is a 2ϕ expander, R overlaps with half of the volume of *every* low conductance cut, i.e. for any cut $(S, \overline{S})$ where $\Phi(S) \leq \phi$ and $\mathbf{vol}(S) \leq \mathbf{vol}(\overline{S})$, we have $\mathbf{vol}(S \cap R) \geq \mathbf{vol}(S)/2$.

The previous best spectral-based algorithm by Orecchia, Sachdeva, and Vishnoi [OSV12] has running time $\tilde{O}(m)$ but the conductance of the output cut can be $\Omega(\sqrt{\phi})$. The previous best flow-based algorithms have running time $\tilde{O}(m)$, and combine the approximate max flow algorithm [Pen16] with the framework by Orecchia et al. [OSVV08] or Sherman [She09]. These algorithms return a cut with conductance $O(\phi \log n)$ which is better than ours. But the running time is $\Omega(m \log^{41} m)$ as they need approximate max flow[7] which is much more complicated than ours.

[6] As shown in [ST04], spectral sparsification on expanders can be done by sampling each edge independently.

[7] The exponent of 41 may have been improved to some smaller (yet still much larger than 5) number due to recent progress in approximate max flow.

Short cycle decomposition. An $(\hat{m}, L)$-*short cycle decomposition* of an undirected graph G, decomposes G into edge-disjoint cycles, each of length at most L, with at most $\hat{m}$ edges not in these cycles. Chu et al. [CGP+18] give a short-cycle decomposition construction using the expander decomposition of Nanongkai and Saranurak [NSW17]. Given an n-node m-edge graph G, they return $O(n^{1+O(1/\log^{1/4} n)}, n^{O((\log\log n)^{3/4}/\log^{1/4} n)})$-short cycle decomposition in time $m^{1+O(1/\log^{1/4} n)}$. Plugging in our new expander decomposition immediately improves the result to:

Corollary 4.3. *There is an algorithm that, given a graph G with with n vertices and m edges, returns a $O(n^{1+O(\log\log n/\sqrt{\log n})}, n^{O(\log\log n/\sqrt{\log n})})$-short cycle decomposition of G in time $m^{1+O(1/\sqrt{\log n})}$.*

Chu et al. [CGP+18] show several applications of short cycle decomposition including degree-preserving spectral sparsification, sparsification of Eulerian directed graphs, and graphical spectral sketches and resistance sparsifiers. Corollary 4.3 implies algorithms for constructing all these objects in the same running time.

Dynamic minimum spanning forests. Expanders are well connected and are "robust" under edge deletions (i.e. k edge deletions can disconnect only $\tilde{O}(k/\phi)$ volume of a graph), and thus many dynamic graph problems become easier once we assume that the underlying graph is always an expander (see e.g. [PT07]). A recent development on dynamic (minimum) spanning trees [NS17, Wul17, NSW17] uses expander decomposition to decompose a graph into expanders and expander pruning to maintain such an expander under edge deletions. With this approach, they improved the long-standing $O(\sqrt{n})$ worst-case update time on an n-node graph of [Fre85, EGIN97] to $n^{O(\log\log\log n/\log\log n)} = n^{o(1)}$.

As we improve the running time of both expander decomposition and pruning in Theorem 1.2 and Theorem 1.3 respectively, we immediately improve the $o(1)$ in their running time as follows:

Corollary 4.4. *There is a Las Vegas randomized dynamic minimum spanning forest algorithm on an n-node graph undergoing edge insertions and deletions with $O(n^{O(\sqrt{\log\log n/\log n})})$ worst-case update time both in expectation and with high probability.*

Dynamic low diameter decomposition. A (β, D) low diameter decomposition of an n-node m-edge graph $G = (V, E)$ is a partition $V_1, \ldots, V_k$ of V into clusters where, for each i, the induced subgraph $G[V_i]$ has diameter at most D and the number of inter-cluster edges is at most βm. Goranci and Krinninger [GK18] show a dynamic algorithm for maintaining a low diameter decomposition which implies dynamic low-stretch spanning trees as an application.

From our new expander decomposition and pruning algorithm, we can maintain low diameter decomposition with almost the same guarantees as in [GK18] (up to a small polylogarithmic factor). But we do not need to assume that the adversary is *oblivious* (i.e. the adversary fixes the whole sequence of updates from the beginning)[8].

Corollary 4.5. *Given any unweighted, undirected multigraph undergoing edge insertions and deletions, there is a fully dynamic algorithm for maintaining a $(\beta, \tilde{O}(1/\beta))$ low diameter decomposition that has amortized update time $\tilde{O}(1/\beta^2)$. The amortized number of edges to become inter-cluster edges after each update is $\tilde{O}(1/\beta)$. These guarantees hold for a* non-oblivious *adversary.*

While many randomized dynamic algorithms in the literature assume an oblivious adversary (e.g. [BGS11, BKS12, KKM13, HKN14a, HKN14b, GK18]), there are very few techniques in dynamic graph algorithm for removing the oblivious adversary assumption. This is important for some applications and is also a stepping stone for derandomizing such algorithms. Corollary 4.5 can be seen as a progress in this research program.

[8]This result, however, does not imply dynamic low-stretch spanning tree algorithm without the oblivious adversary assumption, because there are other steps which need this assumption.

4.3 Open problems

Derandomizing the expansion decomposition in Theorem 1.2 is very interesting as it would break a long-standing $O(\sqrt{n})$ worst-case update time of deterministic dynamic connectivity on n-node graphs to $n^{o(1)}$ worst-case update time by using the framework of [NSW17]. An efficient parallel or distributed implementation of our expander decomposition algorithm will also be of interesting. Chang, Pettie, and Zhang [CPZ18] recently give a distributed algorithm of a weaker variant of expander decomposition with some application.

For the trimming step, can we remove the dependency on $1/\phi$ in the running time? This will very likely require some dynamic version of the nearly linear time approximate max flow algorithms [KLOS14, She13]. To what extent can we reduce the number of inter cluster edges? Improving ours by a logarithmic factor to $O(\phi m \log^2 m)$ might be possible using a more complicated variant of the cut-matching game [OSVV08]. A challenging question about expander pruning algorithm in Theorem 1.3 is whether the amortized update time of $O(\log n/\phi^2)$ can be made worst-case[9].

Acknowledgement

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672. Saranurak was also partially supported by the Swedish Research Council (Reg. No. 2015-04659.)

References

[AALG18] Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. Graph clustering using effective resistance. In *ITCS*, volume 94 of *LIPIcs*, pages 41:1–41:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. 1

[ACK+16] Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P. Woodruff, and Qin Zhang. On sketching quadratic forms. In *ITCS*, pages 311–319. ACM, 2016. 1, 3

[ACL06] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486. IEEE Computer Society, 2006. , 3

[AL08] Reid Andersen and Kevin J. Lang. An algorithm for improving graph partitions. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 651–660, 2008. 8

[AP09] Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. In *STOC*, pages 235–244. ACM, 2009. 3

[ARV09] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2), 2009. 2

[BGS11] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in O (log n) update time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 383–392, 2011. 16

[9] Currently, using Theorem 1.3 and the technique from Section 5.2 of [NSW17], we can obtain a worst-case algorithm with bad update time and guarantee. More precisely, when initially $\phi = \Omega(1/\text{polylog}(n))$, the update time is $2^{O(\sqrt{\log n})}$. More importantly, it only guarantees that the set P contains some set W where $G\{V - W\}$ is a $1/2^{O(\sqrt{\log n})}$ expander, instead of the complement $G\{V - P\}$ itself

[BKS12] Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35, 2012. 16

[CGP+18] Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. *CoRR*, abs/1805.12051, 2018. 1, 2, 3, 16

[Che69] Jeff Cheeger. A lower bound for the smallest eigenvalue of the Laplacian. In *Proceedings of the Princeton conference in honor of Professor S. Bochner*, 1969. 2

[CKP+17] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In *STOC*, pages 410–419. ACM, 2017. 1, 2, 3, 15

[CPZ18] Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. *CoRR*, abs/1807.06624, 2018. To appear in SODA'19. 17

[EGIN97] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. Announced at FOCS 1992. 3, 16

[ES81] Shimon Even and Yossi Shiloach. An On-Line Edge-Deletion Problem. *Journal of the ACM*, 28(1):1–4, 1981. 4

[For10] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75 – 174, 2010. 1

[Fre85] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. Announced at STOC'83. 3, 16

[GK18] Gramoz Goranci and Sebastian Krinninger. Dynamic low-stretch trees via dynamic low-diameter~decompositions. *CoRR*, abs/1804.04928, 2018. 16

[Gol84] A. V. Goldberg. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 1984. 8

[GT88] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, October 1988. 10, 21

[GT12] Shayan Oveis Gharan and Luca Trevisan. Approximating the expansion profile and almost optimal local graph clustering. In *FOCS*, pages 187–196. IEEE Computer Society, 2012. 3

[HdLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. Announced at STOC 1998. 4

[HK99] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. Announced at STOC 1995. 4

[HKN14a] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 18-21, 2014, Philadelphia, PA, USA*, pages 146–155, 2014. 16

[HKN14b] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *STOC*, pages 674–683, 2014. 16

[HRW17] Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1919–1938, 2017. 4, 10, 11, 14, 21, 27

[JS18] Arun Jambulapati and Aaron Sidford. Efficient $\tilde{O}(n/\epsilon)$ spectral sketches for the laplacian and its pseudoinverse. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2487–2503, 2018. 1, 3, 15

[KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142. SIAM, 2013. 16

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In Chandra Chekuri, editor, *SODA*, pages 217–226. SIAM, 2014. 1, 2, 10, 15, 17

[KRV09] Rohit Khandekar, Satish Rao, and Umesh V. Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4), 2009. 2, 5, 27, 28

[KVV00] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings - good, bad and spectral. In *FOCS*, pages 367–377. IEEE Computer Society, 2000. , 1, 2

[LR04] Kevin J. Lang and Satish Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *IPCO*, volume 3064 of *Lecture Notes in Computer Science*, pages 325–337. Springer, 2004. 8

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014. 9

[Mad16] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 593–602, 2016. 9

[NS17] Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o($n^{1/2 - \epsilon}$)-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1122–1129, 2017. , 1, 2, 3, 4, 8, 16

[NSW17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, pages 950–961. IEEE Computer Society, 2017. 1, 2, 3, 4, 8, 16, 17

[OA14] Lorenzo Orecchia and Zeyuan Allen 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*, pages 1267–1286, 2014. 4, 8, 10, 11, 27

[OSV12] Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi. Approximating the exponential, the lanczos method and an $\tilde{o}(m)$-time spectral algorithm for balanced separator. In *STOC*, pages 1141–1160. ACM, 2012. 3, 4, 15

[OSVV08] Lorenzo Orecchia, Leonard J. Schulman, Umesh V. Vazirani, and Nisheeth K. Vishnoi. On partitioning graphs via single commodity flows. In *STOC*, pages 461–470. ACM, 2008. 2, 15, 17

[OV11] Lorenzo Orecchia and Nisheeth K. Vishnoi. Towards an sdp-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition. In *SODA*, pages 532–545. SIAM, 2011. 3, 4

[Pen16] Richard Peng. Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time. In *SODA*, pages 1862–1867. SIAM, 2016. 2, 15

[PT07] Mihai Patrascu and Mikkel Thorup. Planning for fast connectivity updates. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 263–271, 2007. 16

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *SODA*, pages 227–238. SIAM, 2014. 5, 27

[Sch07] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007. 1

[She09] Jonah Sherman. Breaking the multicommodity flow barrier for o(vlog n)-approximations to sparsest cut. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 363–372, 2009. 2, 15

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 263–269, 2013. 10, 17

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000. 1

[ST83] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. 28

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90. ACM, 2004. , 1, 2, 3, 15

[ST11] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011. 2

[ST13] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013. , 4

[Tre05] Luca Trevisan. Approximation algorithms for unique games. In *FOCS*, pages 197–205. IEEE Computer Society, 2005. 1, 3

[VGM16] Nate Veldt, David F. Gleich, and Michael W. Mahoney. A simple and strongly-local flow-based method for cut improvement. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1938–1947. JMLR.org, 2016. 8

[WFH+17] Di Wang, Kimon Fountoulakis, Monika Henzinger, Michael W. Mahoney, and Satish Rao. Capacity releasing diffusion for speed and locality. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3598–3607. PMLR, 2017. 8

[Wul17] Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1130–1143, 2017. , 1, 2, 3, 4, 16

A Details of *Unit-Flow*

Our trimming step exploits the *Unit-Flow* algorithm in [HRW17], which is a specific adaptation of the generic Push-Relabel framework in the seminal work of Goldberg and Tarjan ([GT88]). For completeness, we briefly discuss the *Unit-Flow* algorithm in this section for unweighted graphs. The pseudo-code (Algorithm 3) and some of the results are directly from [HRW17]. We refer reader to [HRW17] for more details. We also sketch the proof of Lemma 3.4 and Lemma 3.7.

Unit-Flow is based on the simple idea that if the underlying graph has large conductance, it will be abundant in disjoint short paths, so flow computation using Push-Relabel will be very efficient as we don't need to raise the vertices' labels very high. In our setting, either the underlying graph indeed has large conductance, so *Unit-Flow* finds a feasible routing very efficiently, or it fails and returns only a pre-flow, but in the latter case we can find an approximate tight cut. Note in our trimming step, we may add $2/\phi$ units of initial mass on nodes, which is different from the setting in [HRW17], where each node has initial mass at most twice its degree.

For efficiency, we make two adaptations to the generic Push-Relabel. First, the labels of nodes cannot become larger than the input parameter h. If $l(v)$ is raised to h and we still have $\mathrm{ex}(v) > 0$, we simply leave the excess on v. Formally, a node v is *active* if $\mathrm{ex}(v) > 0$ and $l(v) < h$, and we maintain a queue Q of active nodes. Moreover, at any point of our algorithm, we consider a unit of excess on a node u as *initial excess* if the excess is initially placed on u by $\Delta(\cdot)$ and is never moved by our algorithm (it is helpful to think of mass as distinct discrete tokens). The second adaptation is that through the execution of our algorithm, we will maintain the property that if any excess on a node u is not initial excess, then $f(u) \leq 2\deg(u)$. Note the property holds at the beginning as all excess are initial excess, and we explicitly enforce that our algorithm never pushes excess to a node u if that would result in $f(u) > 2\deg(u)$.

Same as in generic Push-Relabel, at any point through the execution of *Unit-Flow*, an arc (u, v) with positive residual capacity[10] must have $l(u) \leq l(v) + 1$.

We also need to specify how we maintain the list of active vertices Q. We keep all active vertices in non-decreasing order with respect to their labels (breaking ties arbitrarily). Thus, when we access the first vertex v in Q, v is the active node with smallest label, and if $\mathrm{Push}(v, u)$ is called, the assertion $f(u) < 2\deg(u)$ must be satisfied, as otherwise u will have excess on it (i.e. active) and has smaller label. It is easy to see that adding a node to Q, removing a node from Q, or moving the position of a node in Q due to relabel can all be carries out in $O(1)$ time using one linked list for each label value.

We make the following observations about the *Unit-Flow* algorithm.

[10] Recall the residual capacity of an arc (u, v) is $r_f(u, v) = c(u, v) - f(u, v)$ where in our trimming step $c(u, v) = 2/\phi$ for all edges.

Algorithm 3 Unit Flow

Unit-Flow(G,Δ,h)
. **Initialization:**
. . $\forall\{v,u\} \in E$, $f(u,v) = f(v,u) = 0$.
. . $Q = \{v|\Delta(v) > d(v)\}$, $\forall v$, $l(v) = 0$.
. **While** Q is not empty
. . Let v be the first vertex in Q.
. . *Push/Relabel*(v).
. **End While**

Push/Relabel(v)
. **If** $\exists$ arc (v,u) such that
$r_f(v,u) > 0$, $l(v) = l(u) + 1$
. . *Push*(v,u).
. **Else** *Relabel*(v).
. **End If**

Push(v,u)
. **Assertion**: $f(u) < 2\deg(u)$.
. $\psi = \min(\mathrm{ex}(v), r_f(v,u), 2\deg(u) - f(u))$
. Send ψ 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)
. **Assertion**: v is active, and $\forall u \in V$
$$r_f(v,u) > 0 \implies l(v) \leq l(u).$$
. $l(v) \leftarrow l(v) + 1$.

Observation A.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 possible only when $l(v) > 0$ at the time of the push.

(2) Each vertex v is a sink that can absorb up to $\deg(v)$ units of mass, so we call the $\min(f(v), \deg(v))$ units of mass at v the absorbed *mass. The amount of absorbed mass at v is between $[0, \deg(v)]$, and is non-decreasing. Thus any time after the point that v first becomes active, the sink at v is saturated. In particular any time the algorithm relabels v, there must be $\deg(v)$ units of absorbed mass in the sink at v.*

(3) $f(v) \leq \max(2, \omega) \cdot \deg(v)$, where $\omega = \max_v \Delta(v)/\deg(v)$. This is explicitly maintained.

Upon termination of Unit-Flow*, we have*

(4) 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 $\mathrm{ex}(v) > 0$ for some v, we know all such vertices must be on level h, as the algorithm only stops trying to route v's excess to sinks when v reaches level h. Thus we have the following lemma which should be clear from the above observations.

Lemma A.2. *Upon termination of* Unit-Flow *with input (G, Δ, U, h), assuming $\Delta(v) \leq \omega \deg(v)$ for all v, the pre-flow and labels satisfy*

(a) If $l(v) = h$, then $\max(2, \omega) \cdot \deg(v) \geq f(v) \geq \deg(v)$;

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

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

Now we prove Lemma 3.4

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, and we end up with case (1).

As $|\Delta(\cdot)| \leq 3m/2$, we only have enough mass to saturate a region of volume $3m/2$, thus $\mathbf{vol}(B_0) > m/2$. We consider level cuts as follows: Let $S_i = \cup_{j=i}^{h} B_j$ be the set of vertices with labels at least i. 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. $l(v) = i, l(u) = i - 1$.
2. $l(v) - l(u) > 1$. All edge of this type must be saturated in the down-hill direction. That is, U units of mass are routed from S_i to $V \setminus S_i$ along each such edge.

Suppose there are $z_1(i)$ edges of the first type, we sweep from h to 1, and there must be some $i^* \in [1, h]$ such that

$$z_1(i^*) \leq \frac{5\mathbf{vol}(S_{i^*}) \ln m}{h}. \tag{1}$$

We let S be S_{i^*}, and as S is a level cut, S clearly satisfies all the properties in case (2) of the lemma. We only need to argue its existence. By the following region growing argument, we can show there must exist such $i^* \geq 1$. We start the region growing argument from $i = h$ down to 1. By contradiction,

suppose $z_1(i) \geq \frac{5\mathbf{vol}(S_i)\ln m}{h}$ for all $i \in [1,h]$, which implies $\mathbf{vol}(S_i) \geq \mathbf{vol}(S_{i+1})(1+\frac{5\ln m}{h})$ for all $hi \in [1,h)$. Since $\mathbf{vol}(S_h) = \mathbf{vol}(B_h) \geq 1$, we will have $\mathbf{vol}(S_1) \geq (1+\frac{5\ln m}{h})^h \gg 2m$, which contradicts $\mathbf{vol}(S_1) \leq 3m/2$.

If we need to compute a level cut S, the sweep cut procedure takes $O(\mathbf{vol}(S))$ since we stop as soon as we find a satisfiable S. □

Before proving Lemma 3.7, we first look at the simplified case where we run *Unit-Flow* for one flow problem with initial mass given by $\Delta(\cdot)$ and label bound h.

Lemma A.3. *The running time of* Unit-Flow *is* $O(|\Delta(\cdot)|h)$.

Proof. The initialization of $f(v)$'s and Q takes time linear in $|\Delta(\cdot)|$. For the work carried out by *Unit-Flow*, 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 mass we have, so that each unit of mass gets charged $O(h)$ work. This will prove the result, as there are $|\Delta(\cdot)|$ units of mass in total.

In each iteration of *Unit-Flow*, we look at the first element v of Q, suppose $l(v) = i$ at that point. If the call to *Push/Relabel*(v) ends with a push of ψ units of mass, the iteration takes $O(\psi)$ total work. Note we must have $\psi \geq 1$ by the assertion of Push(v,u). We charge the work of the iteration to that push. 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 A.1(1), then we charge the work to the final relabel of v. Note if we cannot push along an arc (v,u) at some point, we won't be able to push along the arc until the label of u raises. Thus, for any fixed label value of v, we only need to go through its list of incident edges once. Consequently, a relabel of v must be incurred when $\deg(v)$ consecutive calls to *Push/Relabel*(v) end with non-push, so each relabel of v gets charged $O(\deg(v))$ by our charging scheme above.

So far we have charged all the work to pushes and relabels, such that a push of ψ units of mass takes $O(\psi)$, and each relabel of v takes $O(\deg(v))$. We now charge the work of pushes and relabels to the mass. We consider the absorbed mass at v as the first up to $\deg(v)$ units of mass started at or pushed into v, and these units never leave v. We call a unit of excess on u initial excess if it started at u as source mass and never moved.

By Observation A.1(2), each time we relabel v, there are $\deg(v)$ units of absorbed mass at v, so we charge the $O(\deg(v))$ work of the relabel to the absorbed mass, and each unit gets charged $O(1)$. A vertex v is relabeled at most h times, so each unit of mass, as absorbed mass, 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 mass decreases Λ by exactly ψ, since ψ units of excess mass 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 $\mathrm{ex}(v)$. If $\mathrm{ex}(v) > \deg(v)$, all the excess on v must be initial excess of v, as we won't allow $f(v) > 2\deg(v)$ when we push mass to v. In this case, we charge $O(1)$ to each unit of excess on v, and in total each unit of mass, as initial excess, can be charged $O(h)$. If $\mathrm{ex}(v) \leq \deg(v)$, we can charge the increase of Λ to the $\deg(v)$ absorbed mass at v, and each unit gets charged with $O(1)$. In total we can charge all pushes (via Λ) to the mass we have, and each unit of mass is charged with $O(h)$ in total as absorbed mass or initial excess. □

Now we prove the running time when we run *Unit-Flow* over rounds in the trimming step (Lemma 3.7).

Proof. The proof is very similar to the proof of Lemma A.3, and we only point out how to change the argument in proof of Lemma A.3 to accommodate the mass added between rounds.

We can still charge all operations to pushes and relabels, and then charge the pushes to the increment of the potential function

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

The only part that differs from Lemma A.3 is that apart from relabels, we may also increase Λ when we add initial mass to nodes between rounds. In this case, Λ can be increased by at most the amount of new mass added multiplied by the label of the node where we add the mass to, thus we can charge each unit of added mass at most $O(h)$ for the increment of Λ. Each unit of mass is only charged once this way when it is added.

Same as in the proof of Lemma A.3, we can then charge all the work to mass, such that each unit of mass is charged with $O(h)$. As the total amount of mass we create across all rounds is $O(|E(A,R)|/\phi)$, the total running time of *Unit-Flow* over all rounds is $O(|E(A,R)|\log m/\phi^2)$ as we use $h = 10\log m/\phi$.

The time to compute the cut S by sweep cut between rounds takes $O(\mathbf{vol}(S))$, and because the total volume of all the cuts we remove is bounded by $|E(A, V-A)|/\phi$, the running time of all the additional operations between *Unit-Flow* computations is $O(|E(A,V-A)|/\phi)$. □

B The Cut-Matching Step

The goal of this section is to prove Theorem 2.2. We will work on the *subdivision graph*[11] of $G = (V, E)$ defined as follows:

Definition B.1 (Subdivision Graph)**.** *Given a graph $G = (V, E)$, its subdivision graph $G_E = (V', E')$ is the graph where we put a split node x_e on each edge $e \in E$ (including the self-loops). Formally, $V' = V \cup X_E$ where $X_E = \{x_e | e \in E\}$, and $E' = \{\{u, x_e\}, \{v, x_e\} | e = \{u, v\} \in E\}$. We will call nodes in X_E the split nodes, and the other nodes in V' the regular nodes.*

Let us define *X_E-commodity flow* as a multi-commodity flow on G_E such that there are $|X_E|$ flow commodities and every split note $x_e \in X_E$ is a source of quantity 1 of its distinct flow commodity. Only for analysis, we consider an *$m \times m$ flow-matrix* $F \in \mathbb{R}_{\geq 0}^{E\times E}$ which encodes information about an X_E-commodity flow. For any two split nodes x_e and x_h, $F(x_e, x_h)$ indicates how much x_h receives the flow commodity from x_e. We say that F is *routable with congestion c*, if there exists a X_E-commodity flow f is such that, simultaneously for every x_e and x_h, we have that x_e can send $F(x_e, x_h)$ flow commodity to x_h, and the amount of flow on each edge is at most c.

Our algorithm is described in Algorithm 4. We are given $G = (V, E)$ with m edges and ϕ. We assume $\phi < 1/(\log^2 m)$, otherwise the theorem is trivial. Initially, we set $A = V \cup X_E, R = \emptyset, T = \Theta(\log^2 m)$ and $c = 1/(\phi T)$. Only for analysis, we implicitly set F as the identity matrix corresponding to the X_E-the commodity flow where each split node x_e has 1 unit of its own commodity. Trivially, F is routable with zero congestion. Then, we proceed in for at most T rounds and stop as soon as $\mathbf{vol}(R) > m/10T = \Omega(m/\log^2 m)$. For each round t, we will *implicitly* update F such that it is routable with congestion at most ct. The operation for updating F will be described explicitly later in Appendix B.1. If such operation returns a cut $S \subset A$ where $\Phi_{G_E\{A\}}(S) \leq 1/c$, then we "move" S from A to R, i.e., $R = R \cup S$ and $A = A - S$.

Remark B.1. We make several remarks for our following discussion.

[11] The reason we work on this graph is that, technically, the cut-matching framework is for certifying that a graph high *expansion* or finding a cut with low expansion. Expansion is slightly different from conductance. An expansion of a cut S in a graph $G = (V, E)$ is $h(S) = \delta(S)/\min\{|S|, |V - S|\}$. An expansion of a graph G is $h_G = \min_{\emptyset \neq S \subset V} h(S)$.

Algorithm 4 Cut Matching

Cut-Matching(G,ϕ)
. Construct subdivision graph G_E
. $A = V \cup X_E, R = \emptyset, T = \Theta(\log^2 m)$,
  $t = 1, c = 1/(\phi T)$.
. Implicitly set F as the identity matrix of size $m \times m$.
. **While vol**$(R) \leq m/10T$ and $t \leq T$
. . $t = t + 1$
. . Implicitly update F so that F
    is still routable with congestion ct.
. . This update operation might return $S \subset A$
    where $\Phi_{G_E\{A\}}(S) \leq 1/c$.
. . **If** S is returned
. . . $R = R \cup S, A = A - S$

1. We only consider subsets of vertices S in the subdivision graph with the following property. S contains the split vertex x_e for any edge e whose endpoints are both in S; since otherwise we can move x_e into S and make $\Phi_{G_E}(S)$ smaller. This property also holds for A and R.

2. For any subset of nodes U in G_E satisfying the property in the previous remark, there is a corresponding subset $U \cap V$ in G. Up to a constant factor of 2, the volume, cut-size and conductance of U in G_E are the same as the volume, cut-size and conductance of $U \cap V$ in G respectively.

By the above remarks, although the following discussion is in G_E, everything translates easily to G. Now, we analyze the algorithm. Let A_t, R_t, F_t denote A, R, F at round t respectively. For each split node x_e, let $F_t(x_e) \in R^E_{\geq 0}$ be the x_e-row of F_t. We call $F_t(x_e)$ a *flow-vector* of x_e. We define a potential function

$$\psi(t) = \sum_{x_e \in A_t} ||F_t(x_e) - \mu_t||_2^2$$

where $\mu_t = \sum_{x_e \in A_t} F_t(x_e)/|X_E \cap A_t|$ is the average flow vector of split nodes remaining in A_t.

Lemma B.2. *For any $t \leq T$, if $\psi(t) \leq 1/16m^2$, then A_t is a ϕ nearly expander.*

Proof. Here we omit the subscript for readability. For each $x_e \in S$, if $\sum_{x_h \in A} F(x_e, x_h) \geq 1/2$, i.e. total x_e-commodity in A is at least $1/2$, then the x_h-coordiate of $\mu_t(x_h) \geq 1/2m$. As $\psi(t) \leq 1/16m^2$, we have that for every $x_g \in A - S$, $F(x_e, x_h) \geq 1/4m$. Since **vol**(R) is small, and **vol**$(S) \leq$ **vol**$(A)/2$, we know **vol**$(A - S)$ is $\Omega(m)$, so x_e sends out a constant fraction of its flow commodity out of S. In the another case, if $\sum_{x_h \in A} F(x_e, x_h) < 1/2$, then x_e sends out a constant fraction of its flow commodity out of S to R. Summing over all $x_e \in S$, the multi-commodity flow f sends a total amount of at least $\Omega($**vol**$(S))$ out of S. Since the congestion of f is at most $ct = t/\phi T \leq 1/\phi$, we have $|E_{G_E}(S, V - S)| \geq \Omega($**vol**$(S)\phi)$. With proper adjustment on constants in our parameters, we get A is a nearly ϕ expander. □

For readers who are familiar with the cut-matching frame-work, the update operation in Algorithm 4 just implements (a variant of) one round the cut-matching frame-work with the below guarantee. The proof is shown in Appendix B.1.

Lemma B.3. *The operation for updating F is each round takes $O(m/(\phi \log m))$ time. After T rounds, it holds that $\psi(t) \leq 1/16m^2$ w.h.p.*

Proof of Theorem 2.2. Observe first that any round t, we have $\Phi_{G_E}(R_t) \leq 1/c = O(\phi \log^2 m)$. This is because $R_t = \cup_{1 \leq t' \leq t} S_t$ and $\Phi_{G_E\{A_t\}}(S) \leq 1/c$. If Algorithm 4 terminates because $\mathbf{vol}(R_t) > m/10T = \Omega(m/\log^2 m)$, then (A_t, R_t) is a balanced cut where $\Phi_{G_E}(R_t) = O(\phi \log^2 m)$ and we obtain the second case of Theorem 2.2. Otherwise, Algorithm 4 terminates at round T and we apply Lemmas B.2 and B.3. If $R = \emptyset$, then we obtain the first case because the whole node set $V \cup X_E$ is a nearly ϕ expander, which means that G_E is a ϕ expander. In the last case, we write $T = c_0 \log^2 m$ for some constant c_0. We have $\Phi_{G_E}(R_t) \leq 1/c = c_0 \phi \log^2 m$ and $\mathbf{vol}(R_t) \leq m/10T = m/(10c_0 \log^2 m)$. Moreover, A_T is a nearly ϕ expander.

Again, all these three cases translate to the three cases in Theorem 2.2 by Remark B.1. By Lemma B.3, the total running time is $T \times O(m/(\phi \log m)) = O(m \log m/\phi)$. This completes the proof of Theorem 2.2.

B.1 Implicitly Updating Multi-commodity Flows

To prove Lemma B.3, we need to describe how to implicitly update F. This corresponds to one round in the cut-matching game from [KRV09, RST14]. In each round t, we first find a "cut" in $G\{A\}$ using Lemmas B.4 and B.5. More precisely, we actually find two disjoint sets $A^l, A^r \subset A$ and not a cut. Then, we run a flow algorithm which tries to routes mass from A^l to A^r. The returned flow defines a "matching" M between nodes A^l and A^r. Finally, M indicates how to implicitly update F so that it is still routable with congestion ct, and, once $t = T$, F certifies that the remaining A is a nearly ϕ expander as Lemma B.3 needs.

Now, we describe the algorithm in details. Suppose we are now at round t. We will update F_t to be F_{t+1}. Let A^l and A^r be constructed by the two lemmas below.

Lemma B.4 (Lemma 3.4 [KRV09])**.** *Let $u_e = \langle F_t(x_e), r \rangle$ be the projection of x_e's flow-vector to a random unit vector r orthogonal to the all-ones vector. We have $\mathbb{E}[(u_e - u_h)^2] = \|F_t(x_e) - F_t(x_h)\|_2^2 / m$ for all pairs x_e, x_h, and*

$$(u_e - u_h)^2 \leq \frac{C \log m}{m} \|F_t(x_e) - F_t(x_h)\|_2^2$$

hold for all pairs with high probability for some constant C.

Note that F is not explicitly maintained. Despite this fact, Lemma B.8 shows that all u_e's can be computed in $O(m \log^2 m)$.

Lemma B.5 (Lemma 3.3 in [RST14])**.** *Given u_e's for all $x_e \in A \cap X_E$, we can find in time $O(|A| \log |A|)$ a set of source split nodes $A^l \subset A$, and a set of target split nodes $A^r \subset A$, and a separation value η such that*

1. *η separates the sets A^l, A^r, i.e., either $\max_{x_e \in A^l} u_e \leq \eta \leq \min_{x_h \in A^r} u_h$ or $\min_{x_e \in A^l} u_e \geq \eta \geq \max_{x_h \in A^r} u_h$,*
2. *$|A^r| \geq |A \cap X_E|/2$, $|A^l| \leq |A \cap X_E|/8$,*
3. *$\forall x_e \in A^l : (u_e - \eta)^2 \geq \frac{1}{9}(u_e - \bar{\mu})^2$, where $\bar{\mu} = \langle \mu_t, r \rangle$ is the projection of average flow vector μ_t on the random direction,*
4. *$\sum_{x_e \in A^l} (u_e - \bar{\mu})^2 \geq \frac{1}{80} \sum_{x_e \in A} (u_e - \bar{\mu})^2$*

Given A^l and A^r, consider a flow problem on $G\{A\}$ where each split nodes in A^l is a source of 1 unit of mass and each split nodes in A^r is a sink with capacity 1 unit. Every edge has the same capacity $U = 1/(\phi \log^2 m)$. This flow problem can be easily solved by push-relabel algorithms or blocking-flow algorithms whose labels are bounded by the parameter $h = 1/(\phi \log m)$ (e.g. *Unit-Flow* [HRW17] or the block-flow algorithm in [OA14]).

Lemma B.6. *In time $O(mh) = O(m/(\phi \log m))$, we can either find*

1. *A feasible flow f for the above flow problem, or*
2. *A cut S where $\Phi_{G\{A\}}(S) = O(\phi \log^2 m)$ and a feasible flow for the above flow problem when only split nodes in $A^l - S$ are sources of 1 unit.*

Let M_t be a (non-perfect) matching from nodes in A^l to nodes in A^r. If a unit of mass from $x_e \in A^l$ is routed through f to $x_h \in A^r$, then M_t contains an edge (x_e, x_h). We can construct M_t from f using, for example, a dynamic tree [ST83] in $O(m \log m)$ time. From the matching M_t, we update F_t to F_{t+1} as follows: for each $(x_e, x_h) \in M_t$

$$F_{t+1}(x_e) = F_t(x_e)/2 + F_t(x_h)/2.$$

Next, we list properties of F_{t+1}.

Lemma B.7. *F_{t+1} is routable with congestion $c(t+1)$.*

Proof. By induction F_t is routable with congestion ct. As f is feasible for the flow problem whose edge capacity is c, it has congestion c. By using the flow path of f to average the commodities, we know that F_{t+1} is routable with congestion $ct + c = c(t+1)$. □

Lemma B.8. *For any m-dimensional vector r, all $u_e = \langle F_t(x_e), r\rangle$ can be computed in $O(mt)$ time.*

Proof. Treat M_t has an $m \times m$ adjacency matrix of itself. Observe that $F_t = (\frac{I+M_t}{2}) \times \cdots \times (\frac{I+M_1}{2})$. So u_e is just the x_e-th entry of this vector: $F_t r$, which can be computed in $O(mt)$ time because M_i has only $O(m)$ non-zeros for each i. □

Lemma B.9 (Lemma 3.3 in [KRV09])**.** *Before removing S from A, the potential function $\psi(t)$ is reduced by at least $\frac{1}{2}\sum_{(x_e,x_h)\in M_t} \|F_t(x_e) - F_t(x_h)\|_2^2$.*

Lemma B.10. *$\psi(t+1)$ is reduced by a $(1-\Omega(1/\log m))$ factor comparing to $\psi(t)$ with high probability.*

Proof. Recall that $\psi(t) = \sum_{x_e \in A} ||F_t(x_e) - \mu_t||_2^2$.

$$\begin{aligned}
&\psi(t) - \psi(t+1)\\
\geq &\sum_{x_e \in S} \|F_t(x_e) - \mu_t\|_2^2 + \frac{1}{2}\sum_{(x_e,x_h)\in M_t} \|F_t(x_e) - F_t(x_h)\|_2^2\\
\geq &\frac{m}{C\log m}\sum_{x_e \in A^l \cap S}(u_e - \bar{\mu})^2 + \frac{m}{2C\log m}\sum_{(x_e,x_h)\in M_t}(u_e - u_h)^2\\
\geq &\frac{m}{C\log m}\sum_{x_e \in A^l \cap S}(u_e - \bar{\mu})^2 + \frac{m}{2C\log m}\sum_{x_e \in A^l \cap (A-S)}(u_e - \eta)^2\\
\geq &\frac{m}{C\log m}\sum_{x_e \in A^l \cap S}(u_e - \bar{\mu})^2 + \frac{m}{18C\log m}\sum_{x_e \in A^l \cap (A-S)}(u_e - \bar{\mu})^2\\
\geq &\sum_{x_e \in A^l}\frac{m}{18C\log m}(u_e - \bar{\mu})^2\\
\geq &\sum_{x_e \in A}\frac{m}{1440C\log m}(u_e - \bar{\mu})^2
\end{aligned}$$

In the first inequality, the first term is from removing S from A in round $t+1$ and the second term is by Lemma B.9. The second inequality follows from Lemma B.4. The third inequality follows by Lemma B.5(1). The fourth and the last inequalities follow Lemma B.5(3) and (4) respectively.

The above bound hold with high probability. Now use Lemma B.4 again, we know the expectation of the potential drop is

$$\begin{aligned}\mathbb{E}[\sum_{x_e \in A} \frac{m}{1440C \log m}(u_e - \bar{\mu})^2] &= \sum_{x_e \in A} \frac{\|f_t(x_e) - \mu_t\|_2^2}{1440C \log m} \\ &= \Omega(\psi(t)/\log m)\end{aligned}$$

□