58th Annual IEEE Symposium on Foundations of Computer Science

Dynamic Minimum Spanning Forest with Subpolynomial Worst-case Update Time

Danupon Nanongkai
KTH Royal Institute of Technology
Sweden

Thatchaphol Saranurak
KTH Royal Institute of Technology
Sweden

Christian Wulff-Nilsen
University of Copenhagen
Denmark

***Abstract*—We present a Las Vegas algorithm for dynamically maintaining a minimum spanning forest of an n-node graph undergoing edge insertions and deletions. Our algorithm guarantees an $O(n^{o(1)})$ *worst-case* update time with high probability. This significantly improves the two recent Las Vegas algorithms by Wulff-Nilsen [2] with update time $O(n^{0.5-\epsilon})$ for some constant $\epsilon > 0$ and, independently, by Nanongkai and Saranurak [3] with update time $O(n^{0.494})$ (the latter works only for maintaining a spanning forest).**

Our result is obtained by identifying the common framework that both two previous algorithms rely on, and then improve and combine the ideas from both works. There are two main algorithmic components of the framework that are newly improved and critical for obtaining our result. First, we improve the update time from $O(n^{0.5-\epsilon})$ in [2] to $O(n^{o(1)})$ for decrementally removing all low-conductance cuts in an expander undergoing edge deletions. Second, by revisiting the "contraction technique" by Henzinger and King [4] and Holm et al. [5], we show a new approach for maintaining a minimum spanning forest in connected graphs with very few (at most $(1 + o(1))n$) edges. This significantly improves the previous approach in [2], [3] which is based on Frederickson's 2-dimensional topology tree [6] and illustrates a new application to this old technique.

***Keywords*-dynamic graph algorithms; minimum spanning forests; graph decomposition;**

I. Introduction

In the *dynamic minimum spanning forest (MSF)* problem, we want to maintain a minimum spanning forest F of an undirected edge-weighted graph G undergoing edge insertions and deletions. In particular, we want to construct an algorithm that supports the following operations.

- PREPROCESS(G): Initialize the algorithm with an input graph G. After this operation, the algorithm outputs a minimum spanning forest F of G.
- INSERT(u, v, w): Insert edge (u, v) of weight w to G. After this operation, the algorithm outputs changes to F (i.e. edges to be added to or removed from F), if any.
- DELETE(u, v): Delete edge (u, v) from G. After this operation, the algorithm outputs changes to F, if any.

The goal is to minimize the *update time*, i.e., the time needed for outputting the changes to F given each edge update. We call an algorithm for this problem a *dynamic MSF algorithm*. Below, we denote respectively by n and m the upper bounds of the numbers of nodes and edges of G, and use $\tilde{O}$ to hide polylog(n) factors.

The dynamic MSF problem is one of the most fundamental dynamic graph problems. Its solutions have been used as a main subroutine for several static and dynamic graph algorithms, such as tree packing value and edge connectivity approximation [7], dynamic k-connectivity certificate [8], dynamic minimum cut [9] and dynamic cut sparsifier [10]. More importantly, this problem together with its weaker variants – *dynamic connectivity* and *dynamic spanning forest (SF)*[1]– have played a central role in the development in the area of dynamic graph algorithms for more than three decades. The first dynamic MSF algorithm dates back to Frederickson's algorithm from 1985 [6], which provides an $O(\sqrt{m})$ update time. This bound, combined with the general sparsification technique of Eppstein et al. from 1992 [8], implies an $O(\sqrt{n})$ update time.

Before explaining progresses after the above, it is important to note that the update time can be categorized into two types: An update time that holds for every single update is called *worst-case update time*. This is to contrast with an *amortized update time* which holds "on average"[2]. Intuitively, worst-case update time bounds are generally more preferable since in some applications, such as real-time systems, hard guarantees are needed to process a request before the next request arrives. The $O(\sqrt{n})$ bound of Frederickson and Eppstein et al. [6], [8] holds in the worst case. By allowing the update time to be *amortized*, this bound was significantly improved: Henzinger and King [11] in 1995 showed Las Vegas randomized algorithms with $O(\log^3 n)$ amortized update time for the dynamic SF. The same authors [12] in 1997 provided an $O(\sqrt[3]{n} \log n)$ amortized update time for the more general case of dynamic MSF. Finally, Holm et al. [5] in 1998 presented deterministic dynamic SF and MSF

The full version of this paper is available as [1] at https://arxiv.org/abs/1708.03962.

[1]The *dynamic SF* problem is the same as the dynamic MSF problem but we only need to maintain some spanning forest of the graph. In the *dynamic connectivity* problem, we need not to explicitly maintain a spanning forest. We only need to answer the query, given any nodes u and v, whether u and v are connected in the graph.

[2]In particular, for any t, an algorithm is said to have an amortized update time of t if, for any k, the total time it spends to process the first k updates (edge insertions/deletions) is at most kt. Thus, roughly speaking an algorithm with a small amortized update time is fast "on average" but may take a long time to respond to a single update.

0272-5428/17 $31.00 © 2017 IEEE
DOI 10.1109/FOCS.2017.92

algorithms with $O(\log^2 n)$ and $O(\log^4 n)$ amortized update time respectively. Thus by the new millennium we already knew that, with amortization, the dynamic MSF problem admits an algorithm with polylogarithmic update time. In the following decade, this result has been refined in many ways, including faster dynamic SF algorithms (see, e.g. [13], [14], [15] for randomized ones and [16] for a deterministic one), a faster dynamic MSF algorithm [17], and an $\Omega(\log n)$ lower bound for both problems [18].

Given that these problems were fairly well-understood from the perspective of amortized update time, many researchers have turned their attention back to the worst-case update time in a quest to reduce gaps between amortized and worst-case update time (one sign of this trend is the 2007 work of Pătraşcu and Thorup [19]). This quest was not limited to dynamic MSF and its variants (e.g. [20], [21], [22], [23], [24]), but overall the progress was still limited and it has become a big technical challenge whether one can close the gaps. In the context of dynamic MSF, the $O(\sqrt{n})$ worst-case update time of [6], [8] has remained the best for decades until the breakthrough in 2013 by Kapron, King and Mountjoy [25] who showed a Monte Carlo randomized algorithm with polylogarithmic worst-case bound for the dynamic connectivity problem (the bound was originally $O(\log^5 n)$ in [25] and was later improved to $O(\log^4 n)$ in [26]). Unfortunately, the algorithmic approach in [25], [26] seems insufficient for harder problems like dynamic SF and MSF[3], and the $O(\sqrt{n})$ barrier remained unbroken for both problems.

It was only very recently that the *polynomial improvement* to the $O(\sqrt{n})$ worst-case update time bound was presented [2], [3][4]. Wulff-Nilsen [2] showed a Las Vegas algorithm with $O(n^{0.5-\epsilon})$ update time for some constant $\epsilon > 0$ for the dynamic MSF problem. Independently, Nanongkai and Saranurak [3] presented two dynamic SF algorithms: one is Monte Carlo with $O(n^{0.4+o(1)})$ update time and another is Las Vegas with $O(n^{0.49306})$ update time. Nevertheless, the large gap between polylogarithmic amortized update time and the best worst-case update time remains.

Our Result.

We significantly reduce the gap by showing the dynamic MSF algorithm with *subpolynomial* ($O(n^{o(1)})$) update time:

Theorem I.1. *There is a Las Vegas randomized dynamic MSF algorithm on an n-node graph that can answer each update in $O(n^{o(1)})$ time both in expectation and with high probability.*

Needless to say, the above result completely subsumes the result in [2], [3]. The $o(1)$ term above hides a $O(\log\log\log n/\log\log n)$ factor.[5] Recall that Las Vegas randomized algorithms always return correct answers and the time guarantee is randomized. Also recall that an event holds *with high probability* (w.h.p.) if it holds with probability at least $1-1/n^c$, where c is an arbitrarily large constant.

Key Technical Contribution and Organization. We prove Theorem I.1 by identifying the common framework behind the results of Nanongkai-Saranurak [3] and Wullf-Nilsen [2] (thereafter NS and WN), and significantly improving some components within this framework. In particular, in retrospect it can be said that at a high level NS [3] and WN [2] share the following three components:

1) *Expansion decomposition:* This component decomposes the input graph into several expanders and the "remaining" part with few ($o(n)$) edges.
2) *Expander pruning:* This component helps maintaining an MSF/SF in expanders from the first component by decrementally removing all low-conductance cuts in an expander undergoing edge deletions.
3) *Dynamic MSF/SF on ultra-sparse graphs:* This components maintains MSF/SF in the "remaining" part obtained from the first component by exploiting the fact that this part has few edges[6].

The key difference is that while NS [3] heavily relied on developing fast algorithms for these components using recent flow techniques (from, e.g., [28], [29]), WN [2] focused on developing a sophisticated way to integrate all components together and used slower (diffusion-based) algorithms for the three components. In this paper we significantly improve algorithms for the second and third components from those in NS [3], and show how to adjust the integration method of WN [2] to exploit these improvements; in particular, the method has to be carefully applied recursively. Below we discuss how we do this in more detail.

(i) Improved expander pruning (Details in Sections III to VI). We significantly improve the running time of the *one-shot* expander pruning algorithm by NS [3][7] and the *dynamic*

[3]Note that the algorithms in [25], [26] actually maintain a spanning forest; however, they cannot output such forest. In particular, [25], [26] assume the so-called *oblivious adversary*. Thus, [25], [26] do not solve dynamic SF as we define here, as we require algorithms to report how the spanning forest changes. See further discussions on the oblivious adversary in [3].

[4]Prior to this, Kejlberg-Rasmussen et al. [27] improved the bound slightly to $O(\sqrt{n(\log\log n)^2/\log n})$ for dynamic SF using word-parallelism. Their algorithm is deterministic.

[5]Note that by starting from an empty graph and inserting one edge at a time, the preprocessing time of our algorithm is clearly $O(m^{1+o(1)})$, where m is the number of edges in the initial graph. However, note further that the $o(1)$ term in our preprocessing time can be slightly reduced to $O(\sqrt{\log\log m/\log m})$ if we analyze the preprocessing time explicitly instead.

[6]For the reader who are familiar with the results in [3] and [2]. The first component are shown in Theorem 4 in [2] and Theorem 5.1 in [3]. The second are shown in Theorem 5 in [2] and Theorem 6.1 in [3]. The third are shown in Theorem 3 from [2] and Theorem 4.2 in [3].

[7]In [3], the authors actually show the local expansion decomposition algorithm which is the same as one-shot expander pruning but it does not only prune the graph but also decompose the graph into components. In retrospect, we can see that it is enough to instead use the one-shot expander pruning algorithm in [3].

expander pruning by WN [2]. For the one-shot case, given a *single batch* of d edge deletions to an expander, the one-shot expander pruning algorithm by NS [3] takes $O(d^{1.5+o(1)})$ time for removing all low-conductance cuts. We improve the running time to $O(d^{1+o(1)})$. To do this, in Section III we first extend a new *local flow-based algorithm*[8] for finding a low-conductance cut by Henzinger, Rao and Wang [30], and then use this extension in Section IV to get another algorithm for finding a *locally balanced sparse (LBS) cut*. Then in Section V-A we apply the reduction from LBS cut algorithms by NS [3] and obtain an improved one-shot expander pruning algorithm.

For the dynamic case, given a *sequence* of edge deletions to an expander, the dynamic expander pruning algorithm by WN [2] dynamically removes all low-conductance cuts and takes $O(n^{0.5-\epsilon})$ time for each update. We improve the update time to $O(n^{o(1)})$. Our algorithm is also arguably simpler and differ significantly because we do not need random sampling as in [2]. To obtain the dynamic expander pruning algorithm, we use many instances of the static ones, where each instance is responsible on finding low-conductance cuts of different sizes. Each instance is called periodically with different frequencies (instances for finding larger cuts are called less frequently). See Section V-B for details.

(ii) Improved dynamic MSF algorithm on "ultra-sparse" graphs (Details in Section VII). We show a new way to maintain dynamic MSF in a graph with few ($o(n)$) "non-tree" edges that can also handle a batch of edge insertions. Both NS and WN [3], [2] used a variant of Frederickson's 2-dimensional topology tree [6] to do this task[9]. In this paper, we change the approach to reduce this problem on graphs with few *non-tree edges* to the same problem on graphs with few *edges* and *fewer nodes*; this allows us to apply recursions later in Section IX. We do this by applying the classic "contraction technique" of Henzinger and King [4] and Holm et al. [5] *in a new way*: This technique was used extensively previously (e.g. [12], [4], [5], [17], [2]) to reduce fully-dynamic algorithms to decremental algorithms (that can only handle deletions). Here, we use this technique so that we can recurse.

In Sections VIII and IX, we take a close look into the integration method in WN [2] which is used to compose the three components. We show that it is possible to replace all the three components with the tools based on flow algorithms from either this paper or from NS [3] instead.

In particular, in Section VIII we consider a subroutine implicit in WN [2], which is built on top of the expansion decomposition algorithm (the first component above). To make the presentation more modular, we explicitly state this subroutine and its needed properties and name it *MSF decomposition* in Section VIII. This subroutine can be used as it is constructed in [2], but we further show that it can be slightly improved if we replace the diffusion-based expansion decomposition algorithm in [2] with the flow-based expansion decomposition by NS [3] in the construction. This leads to a slight improvement in the $o(1)$ term in our claimed $O(n^{o(1)})$ update time.

Then, in Section IX, we combine (using a method in WN [2]) our improved MSF decomposition algorithm (from Section VIII) with our new dynamic expander pruning algorithm and our new dynamic MSF algorithm on ultra-sparse graphs (for the second and third components above). As our new algorithm on ultra-sparse graphs is actually a reduction to the dynamic MSF problem on a smaller graph, we recursively apply our new dynamic MSF algorithm on that graph. By a careful time analysis of our recursive algorithm, we eventually obtain the $O(n^{o(1)})$ update time.

II. Preliminaries

When the problem size is n, we denote $\tilde{O}(f(n)) = O(f(n)\text{polylog}(n))$, for any function f. We denote by $\dot{\cup}$ and $\dot{\bigcup}$ the disjoint union operations. We denote the set minus operation by both $\setminus$ and $-$. For any set S and an element e, we write $S - e = S - \{e\} = S \setminus \{e\}$.

Let $G = (V, E, w)$ be any weighted graph where each edge $e \in E$ has weight $w(e)$. We usually denote $n = |V|$ and $m = |E|$. We also just write $G = (V, E)$ when the weight is clear from the context. We assume that the weights are distinct. For any set $V' \subseteq V$ of nodes, $G[V']$ denotes the subgraph of G induced by V'. We denote $V(G)$ the set of nodes in G and $E(G)$ the set of edges in G. In this case, $V(G) = V$ and $E(G) = E$. Let $\mathsf{MSF}(G)$ denote the minimum spanning tree of G. For any set $E' \subseteq E$, let $end(E')$ be the set of nodes which are endpoints of edges in E'. Sometimes, we abuse notation and treat the set of edges in E' as a graph $G' = (end(E'), E')$ and vice versa. For example, we have $\mathsf{MSF}(E') = \mathsf{MSF}(G')$ and $E - \mathsf{MSF}(G') = E - E(\mathsf{MSF}(G'))$. The set of *non-tree edges* of G are the edges in $E - \mathsf{MSF}(G)$. However, when it is clear that we are talking about a forest F in G, non-tree edges are edges in $E - F$.

A cut $S \subseteq V$ is a set of nodes. A *volume* of S is $vol(S) = \sum_{v \in S} \deg(v)$. The *cut size* of S is denoted by $\delta(S)$ which is the number of edges crossing the cut S. The *conductance* of a cut S is $\phi(S) = \frac{\delta(S)}{\min\{vol(S), vol(V-S)\}}$. The conductance of a graph $G = (V, E)$ is $\phi(G) = \min_{\emptyset \neq S \subset V} \phi(S)$.

Remark II.1 (Local-style input). Whenever a graph G is given to any algorithm A in this paper, we assume that a *pointer* to the adjacency list representing G is given to A. This is necessary for some of our algorithms which are *local* in the sense that they do not even read the whole input graph. Recall that in an adjacency list, for each node v we have a

[8] By local algorithms, we means algorithms that can output its answer without reading the whole input graph.

[9] Unlike [2], the algorithm in [3] cannot handle inserting a batch of many non-tree edges.

list ℓ_v of edges incident to v , and we can access the head ℓ_v in constant time. (See details in, e.g., [31, Section 22.1]) Additionally, we assume that we have a list of nodes whose degrees are at least 1 (so that we do not need to probe lists of single nodes).

We extensively use the following facts about MSF.

Fact II.2 ([8])**.** *For any edge sets E_1 and E_2, $\mathit{MSF}(E_1 \cup E_2) \subseteq \mathit{MSF}(E_1) \cup \mathit{MSF}(E_2)$.*

Let $G' = (V', E')$ be a graph obtained from G by contracting some set of nodes into a single node. We always keeps parallel edges in G' but sometimes we do not keep all the self loops. We will specify which self loops are preserved in G' when we use contraction in our algorithms. We usually assume that each edge in G' "remember" its original endpoints in G. That is, there are two-way pointers from each edge in E' to its corresponding edge in E. So, we can treat E' as a subset of E. For example, for a set $D \subseteq E$ of edges in G, we can write $E' - D$ and this means the set of edges in G' excluding the ones which are originally edges in D. With this notation, we have the following fact about MSF:

Fact II.3. *For any graph G and (multi-)graph G' obtained from G by contracting two nodes of G, $\mathit{MSF}(G') \subseteq \mathit{MSF}(G)$.*

Definition II.4 (Dynamic MSF)**.** *A* (fully) dynamic MSF algorithm $\mathcal{A}$ *is given an initial graph G to be preprocessed, and then $\mathcal{A}$ must return an initial minimum spanning forest. Then there is an online sequence of edge updates for G, both insertions and deletions. After each update, $\mathcal{A}$ must return the list of edges to be added or removed from the previous spanning tree to obtain the new one. We say $\mathcal{A}$ is an* incremental/decremental *MSF algorithm if the updates only contain insertions/deletions respectively.*

The time an algorithm uses for preprocessing the initial graph and for updating a new MSF is called *preprocessing time* and *update time* respectively. In this paper, we consider the problem where the update sequence is generated by an adversary[10]. We say that an algorithm has *update time t with probability p*, if, *for each* update, an algorithm need at most t time to update the MSF with probability at least p.

Let G be a graph undergoing a sequence of edge updates. If we say that G has n nodes, then G has n nodes at any time. However, we say that G has at most m edges and k non-tree edges, if at any time, G is updated in such a way that G always has at most m edges and k non-tree edges. We also say that G is an *m-edge k-non-tree-edge graph*. Let $F = \mathsf{MSF}(G)$. Suppose that there is an update that deletes $e \in F$. We say that f is a *replacement/reconnecting edge* if $F \cup f - e = \mathsf{MSF}(G - e)$. We also use the the top tree data structure (see e.g. [32], [33]).

[10]There are actually two kinds of adversaries: oblivious ones and adaptive ones. In [3], they formalize these definitions precisely and discuss them in details. In this paper, however, we maintain MSF which is uniquely determined by the underlying graph at any time (assuming that the edge weights are distinct). So, there is no difference in power of the two kinds of adversaries and we will not distinguish them.

III. The Extended Unit Flow Algorithm

In this section, we show an algorithm called *Extended Unit Flow* in Theorem III.3. It is the main tool for developing an algorithm in Section IV called *locally balanced sparse cut*, which will be used in our dynamic algorithm. The theorem is based on ideas of flow algorithms by Henzinger, Rao and Wang [30].

Flow-related notions.

We derive many notations from [30], but note that they are not exactly the same. (In particular, we do not consider edge capacities, but instead use the notion of congestion.) A flow is defined on an instance $\Pi = (G, \Delta, T)$ consisting of (i) an unweighted undirected graph $G = (V, E)$, (ii) a *source function* $\Delta : V \to \mathbb{Z}_{\geq 0}$, and (iii) a *sink function* $T : V \to \mathbb{Z}_{\geq 0}$. A *preflow* is a function $f : V \times V \to \mathbb{Z}$ such that $f(u, v) = -f(v, u)$ for any $(u, v) \in V \times V$ and $f(u, v) = 0$ for every $(u, v) \notin E$. Define $f(v) = \Delta(v) + \sum_{u \in V} f(u, v)$. A preflow f is said to be *source-feasible* (respectively *sink-feasible*) if, for every node v, $\sum_u f(v, u) \leq \Delta(v)$ (respectively $f(v) \leq T(v)$.). If f is both source- and sink-feasible, then we call it a *flow*. We define $\mathit{cong}(f) = \max_{(u,v) \in V \times V} f(u, v)$ as *the congestion of f*. We emphasize that the input and output functions considered here (i.e. Δ, T, f, and cong) map to *integers*.

One way to view a flow is to imagine that each node v initially has $\Delta(v)$ units of *supply* and an ability to *absorb* $T(v)$ units of supply. A preflow is a way to "route" the supply from one node to another. Intuitively, in a valid routing the total supply out of each node v should be at most its initial supply of $\Delta(v)$ (source-feasibility). A flow describes a way to route such that all supply can be absorbed (sink feasibility); i.e. in the end, each node v has at most $T(v)$ units of supply. The congestion measures how much supply we need to route through each edge.

With the view above, we call $f(v)$ (defined earlier) *the amount of supply ending at v after f*. For every node v, we denote $ex_f(v) = \max\{f(v) - T(v), 0\}$ as *the excess supply at v after f* and $ab_f(v) = \min\{T(v), f(v)\}$ as *the absorbed supply at v after f*. Observe that $ex_f(v) + ab_f(v) = f(v)$, for any v, and f is a feasible flow iff $ex_f(v) = 0$ for all nodes $v \in V$. When f is clear from the context, we simply use ex and ab to denote ex_f and ab_f. For convenience, we denote $|\Delta(\cdot)| = \sum_v \Delta(v)$ as *the total source supply*, $|T(\cdot)| = \sum_v T(v)$ as *the total sink capacity*, $|ex_f(\cdot)| = \sum_v ex_f(v)$ as *the total excess*, and $|ab_f(\cdot)| = \sum_v ab_f(v)$ as *the total supply absorbed*.

Remark III.1 (Input and output formats). The input graph G is given to our algorithms as described in Section II;

in particular, our algorithms do not need to read G entirely. Functions Δ and T are input in the form of sets $\{(v, \Delta(v)) \mid \Delta(v) > 0\}$ and $\{(v, T(v)) \mid T(v) < \deg(v)\}$, respectively. Our algorithms will read both sets entirely.

Our algorithms output a preflow f as a set $\{((u,v), f(u,v)) \mid f(u,v) \neq 0\}$. When f is outputted, we can assume that we also obtained functions ex_f and ab_f which are represented as sets $\{(v, ex_f(v)) \mid ex_f(v) > 0\}$ and $\{(v, ab_f(v)) \mid ab_f(v) > 0\}$, respectively. This is because the time for computing these sets is at most linear in the time for reading Δ and T plus the time for outputting f.

Remark III.2 ($\overline{T}(\cdot)$). We need another notation to state our result. Throughout, we only consider sink functions T such that $T(v) \leq \deg(v)$ for all nodes $v \in V$. When we compute a preflow, we usually add to each node v an *artificial supply* $\overline{T}(v) = \deg(v) - T(v)$ to both $\Delta(v)$ and $T(v)$ so that $T(v) = \deg(v)$. Observe that adding the artificial apply does not change the problem (i.e. a flow and preflow is feasible in the new instance if and only if it is in the old one). We define $|\overline{T}(\cdot)| = \sum_v \overline{T}(v) = 2m - |T(\cdot)|$ as the *total artificial supply*. This term will appear in the running time of our algorihtm.

The main theorem.

Now, we are ready to state the main result of this section.

Theorem III.3 (Extended Unit Flow Algorithm)**.** *There exists an algorithm* called Extended Unit Flow *which takes the followings as input:*

- *a graph $G = (V, E)$ with m edges (possibly with parallel edges but without self loops),*
- *positive integers $h \geq 1$ and $F \geq 1$,*
- *a source function Δ such that $\Delta(v) \leq F \deg(v)$ for all $v \in V$, and*
- *a sink function T such that $|\Delta(\cdot)| \leq |T(\cdot)|$ (also recall that $T(v) \leq \deg(v)$, $\forall v \in V$, as in Remark III.2).*

In time $O(hF(|\Delta(\cdot)| + |\overline{T}(\cdot)|) \log m)$ the algorithm returns (i) a source-feasible preflow f with congestion $cong(f) \leq 2hF$ and (ii) $|ex_f(\cdot)|$. Moreover, either

- *(Case 1) $|ex_f(\cdot)| = 0$, i.e. f is a flow, or*
- *(Case 2) the algorithm returns a set $S \subseteq V$ such that $\phi_G(S) < \frac{1}{h}$ and $vol(S) \geq \frac{|ex_f(\cdot)|}{F}$. (All nodes in S are outputted.)*

Interpretation of Theorem III.3.

One way to interpret Theorem III.3 is the following. (Note: readers who already understand Theorem III.3 can skip this paragraph.) Besides graph G and source and sink functions, the algorithm in Theorem III.3 takes integers h and F as inputs. These integers indicate the input that we consider "good": (i) the source function Δ is not too big at each node, i.e. $\forall v \in V,\ \Delta(v) \leq F \deg(v)$, and (ii) the graph G has high conductance; i.e. $\phi(G) > 1/h$. Note that for the good input it is possible to find a flow of congestion $\tilde{O}(hF)$: each set $S \subseteq V$ there can be $\sum_{v \in S} \Delta(v) \leq F \cdot vol(S)$ initial supply (by (i)), while there are $\delta(S) > vol(S)/h$ edges to route this supply out of S (by (ii)); so, on average there is $\frac{\sum_{v \in S} \Delta(v)}{\delta(S)} \leq hF$ supply routed through each edge. This is essentially what our algorithm achieves in Case 1. If it does not manage to compute a flow, it computes some source-feasible preflow and outputs a "certificate" that the input is bad, i.e. a low-conductance cut S as in Case 2. Moreover, the larger the excess of the preflow, the higher the volume of S; i.e. $vol(S)$ is in the order of $ex_f(\cdot)/F$. In fact, this volume-excess relationship is the key property that we will need later. One way to make sense of this relationship is to notice that if $vol(S) \geq |ex_f(\cdot)|/F$, then we can put as much as $F \cdot vol(S) \geq |ex_f(\cdot)|$ initial supply in S. With conductance of S low enough ($\phi_G(S) \leq \frac{1}{2h}$ suffices), we can force most of the initial supply to remain in S and become an excess. Note that this explanation is rather inaccurate, but might be useful to intuitively understand the interplay between $vol(S)$, $|ex_f(\cdot)|$ and F.

Finally, we note again that our algorithm is *local* in the sense that its running time is lower than the size of G. For this algorithm to be useful later, it is important that the running time is *almost-linear* in $(|\Delta(\cdot)| + |\overline{T}(\cdot)|)$. Other than this, it can have any polynomial dependency on h, F and the logarithmic terms.

The main idea for proving Theorem III.3 is to slightly extend the algorithm called *Unit Flow* by Henzinger, Rao and Wang [30]. See the full version of paper for the proof.

IV. Locally Balanced Sparse Cut

In this section, we show an algorithm for finding a *locally balanced sparse cut*, which is a crucial tool in Section V. The main theorem is Theorem IV.4. First, we need this definition:

Definition IV.1 (Overlapping)**.** *For any graph $G = (V, E)$, set $A \subset V$, and real $0 \leq \sigma \leq 1$, we say that a set $S \subset V$ is* (A, σ)-overlapping *in G if $vol(S \cap A)/vol(S) \geq \sigma$.*

Let $G = (V, E)$ be a graph. Recall that a cut S is *α-sparse* if it has conductance $\phi(S) = \frac{\delta(S)}{\min\{vol(S), vol(V-S)\}} < \alpha$. Consider any set $A \subset V$, an overlapping parameter $0 \leq \sigma \leq 1$ and a conductance parameter $0 \leq \alpha \leq 1$. Let S^* be the set of largest volume that is α-sparse (A, σ)-overlapping and such that $vol(S^*) \leq vol(V - S^*)$. We define $\mathsf{OPT}(G, \alpha, A, \sigma) = vol(S^*)$. If S^* does not exist, then we define $\mathsf{OPT}(G, \alpha, A, \sigma) = 0$. From this definition, observe that $\mathsf{OPT}(G, \alpha, A, \sigma) \leq \mathsf{OPT}(G, \alpha', A, \sigma)$ for any $\alpha \leq \alpha'$. Now, we define the locally balanced sparse cut problem formally:

Definition IV.2 (Locally Balanced Sparse (LBS) Cut)**.** *Consider any graph $G = (V, E)$, a set $A \subset V$, and parameters*

$c_{size} \geq 1, c_{con} \geq 1, \sigma$ *and* α*. We say that a cut* S *where* $vol(S) \leq vol(V - S)$ *is a* (c_{size}, c_{con})-approximate locally balanced sparse cut with respect to (G, α, A, σ) *(in short,* $(c_{size}, c_{con}, G, \alpha, A, \sigma)$-LBS cut*) if*

$$\phi(S) < \alpha \qquad \text{and} \qquad c_{size} \cdot vol(S) \geq \mathsf{OPT}(G, \alpha/c_{con}, A, \sigma). \tag{1}$$

In words, the $(c_{size}, c_{con}, G, \alpha, A, \sigma)$-LBS cut can be thought of as a relaxed version of $\mathsf{OPT}(G, \alpha, A, \sigma)$: On the one hand, we define $\mathsf{OPT}(G, \alpha, A, \sigma)$ to be a *highest-volume* cut with low enough conductance and high enough overlap with A (determined by α and σ respectively). On the other hand, a $(c_{size}, c_{con}, G, \alpha, A, \sigma)$-LBS cut does *not* need to overlap with A at all; moreover, its volume is only compared to $\mathsf{OPT}(G, \alpha/c_{con}, A, \sigma)$, which is at most $\mathsf{OPT}(G, \alpha, A, \sigma)$, and we also allow the gap of c_{size} in such comparison. We note that the existence of a $(c_{size}, c_{con}, G, \alpha, A, \sigma)$-LBS cut S implies that any (A, σ)-overlapping cut of volume more than $c_{size} \cdot vol(S)$ must have conductance at least α/c_{con} (because any (A, σ)-overlapping cut with conductance less than α/c_{con} has volume at most $c_{size} \cdot vol(S)$).

Definition IV.3 (LBS Cut Algorithm)**.** *For any parameters* c_{size} *and* c_{con}*,* a (c_{size}, c_{con})-approximate algorithm *for the LBS cut problem (in short,* (c_{size}, c_{con})-approximate LBS cut algorithm*) takes as input a graph* $G = (V, E)$*, a set* $A \subset V$*, an overlapping parameter* $0 \leq \sigma \leq 1$*, and an conductance parameter* $0 \leq \alpha \leq 1$*. Then, the algorithm either*

- *(Case 1) finds a* $(c_{size}, c_{con}, G, \alpha, A, \sigma)$*-LBS cut* S*, or*
- *(Case 2) reports that there is no* (α/c_{con})*-sparse* (A, σ)*-overlapping cut, i.e.* $\mathsf{OPT}(G, \alpha/c_{con}, A, \sigma) = 0$*.*

From Definition IV.3, if there exists an (α/c_{con})-sparse (A, σ)-overlapping cut, then a (c_{size}, c_{con})-approximate LBS cut algorithm $\mathcal{A}$ can only do Case 1, or if there is no α-sparse cut, then $\mathcal{A}$ must do Case 2. However, if there is no (α/c_{con})-sparse (A, σ)-overlapping cut but there is an α-sparse cut, then $\mathcal{A}$ can either do Case 2, or Case 1 (which is to find any α-sparse cut in this case).

The main result of this section is the following:

Theorem IV.4. *Consider the special case of the LBS cut problem where the input (G, A,* σ*,* α*) is always such that (i)* $2vol(A) \leq vol(V - A)$ *and (ii)* $\sigma \in [\frac{2vol(A)}{vol(V-A)}, 1]$*. In this case, there is a* $(O(1/\sigma^2), O(1/\sigma^2))$*-approximate LBS cut algorithm that runs in* $\tilde{O}(\frac{vol(A)}{\alpha\sigma^2})$ *time.*

We note that in our later application it is enough to have an algorithm with poly$(\frac{\log n}{\alpha\sigma})$ approximation guarantees and running time almost linear in $vol(A)$ (possibly with poly$(\frac{\log n}{\alpha\sigma})$).

Before proving the above theorem, let us compare the above theorem to related results in the literature. Previously, Orecchia and Zhu [29] show two algorithms for a problem called *local cut improvement*. This problem is basically the same as the LBS cut problem except that there is no guarantee about the volume of the outputted cut. Nanongkai and Saranurak [3] show that one of the two algorithms by [29] implies a $(\frac{3}{\sigma}, \frac{3}{\sigma})$-approximate LBS cut algorithm with running time $\tilde{O}((\frac{vol(A)}{\sigma})^{1.5})$. While the approximation guarantees are better than the one in Theorem IV.4, this algorithm is too slow for us. By the same techniques, one can also show that the other algorithm by [29] implies a $(n, \frac{3}{\sigma})$-approximate LBS cut algorithm with running time $\tilde{O}(\frac{vol(A)}{\alpha\sigma})$ similar to Theorem IV.4, but the approximation guarantee on c_{size} is too high for us. Thus, the main challenge here is to get a good guarantee on both c_{size} and running time. Fortunately, given the Extended Unit Flow algorithm from Section III, it is not hard to obtain Theorem IV.4. See the full version of the paper for the proof of Theorem IV.4.

V. Expander Pruning

The main result of this section is the *dynamic expander pruning* algorithm. This algorithm was a key tool introduced by Wulff-Nilsen [2, Theorem 5] for obtaining his dynamic MSF algorithm. We significantly improve his dynamic expander pruning algorithm which is randomized and has $n^{0.5-\epsilon_0}$ update time for some constant $\epsilon_0 > 0$. Our algorithm is *deterministic* and has $n^{o(1)}$ update time. Although the algorithm is deterministic, our final dynamic MSF algorithm is randomized because there are other components that need randomization.

First we state the precise statement (explanations follow below).

Theorem V.1 (Dynamic Expander Pruning)**.** *Consider any* $\epsilon(n) = o(1)$*, and let* $\alpha_0(n) = 1/n^{\epsilon(n)}$*. There is a dynamic algorithm* $\mathcal{A}$ *that can maintain a set of nodes* P *for a graph* G *undergoing* $T = O(m\alpha_0^2(n))$ *edge deletions as follows. Let* G_τ *and* P_τ *be the graph* G *and set* P *after the* τ^{th} *deletion, respectively.*

- *Initially, in* $O(1)$ *time* $\mathcal{A}$ *sets* $P_0 = \emptyset$ *and takes as input an* n*-node* m*-edge graph* $G_0 = (V, E)$ *with maximum degree* 3*.*
- *After the* τ^{th} *deletion,* $\mathcal{A}$ *takes* $n^{O(\log\log\frac{1}{\epsilon(n)}/\log\frac{1}{\epsilon(n)})} = n^{o(1)}$ *time to report nodes to be added to* $P_{\tau-1}$ *to form* P_τ *where, if* $\phi(G_0) \geq \alpha_0(n)$*, then*

$$\exists W_\tau \subseteq P_\tau \text{ s.t. } G_\tau[V - W_\tau] \text{ is connected.} \tag{2}$$

The goal of our algorithm is to gradually mark nodes in a graph $G = (V, E)$ so that at all time – as edges in G are deleted – all nodes that are not yet marked are in the same connected component in G. In other words, the algorithm maintains a set P of (marked) nodes, called *pruning set*, such that there exists $W \subseteq P$ where $G[V - W]$ is connected (thus Equation (2)). In our application in Section IX, we will

delete edges incident to P from the graph, hence the name pruning set.

Recall that the algorithm takes an input graph in the local manner, as noted in Remark II.1, thus taking $n^{o(1)}$ time. Observe that if we can set $P = V$ from the beginning, the problem becomes trivial. The challenge here is that we must set $P = \emptyset$ in the initial step, and thus must grow P smartly and quickly (in $n^{o(1)}$ time) after each deletion so that Equation (2) remains satisfied.

Observe further that this task is not possible to achieve in general: if the first deletion cuts G into two large connected components, then P has to grow tremendously to contain one of these components, which is impossible to do in $n^{o(1)}$ time. Because of this, our algorithm is guaranteed to work only if the initial graph has *high enough expansion*; in particular, an initial expansion of $\alpha_0(n)$ as in Theorem V.1 suffices for us.

Organization. The rest of this section is for proving Theorem V.1. The key tool is an algorithm called the *one-shot expander pruning*, which was also the key tool in Nanongkai and Saranurak [3] for obtaining their Las Vegas dynamic SF algorithm. We show an improved version of this algorithm in Section V-A using the faster LBS cut algorithm we developed in Section IV. In Section V-B, we show how to use several instances of the one-shot expander pruning algorithm to obtain the dynamic one and prove Theorem V.1.

A. One-shot Expander Pruning

In the following, we show the *one-shot expander pruning* algorithm which is significantly improved from [3]. In words, the one-shot expander pruning algorithm is different from the dynamic one from Theorem V.1 in two aspects: 1) it only handles a single batch of edge deletions, instead of a sequence of edge deletions, and so only outputs a pruning set P once, and 2) the pruning set P has a stronger guarantee than the pruning set for dynamic one as follows: P does not only contains all nodes in the cuts that are completely separated from the graphs (i.e. the separated connected components) but P contains all nodes in the cuts that have low conductance. Moreover, P contains *exactly* those nodes and hence the complement $G[V - P]$ has high conductance. For the dynamic expander pruning algorithm, we only have that there is some $W \subseteq P$ where $G[V - W]$ is connected.

The theorem below shows the precise statement. Below, we think of $G_b = (V, E \cup D)$ as the graph before the deletions, and $G = G_b - D$ as the graph after deleting D. In [3], Nanongkai and Saranurak show this algorithm where the dependency on D is $\sim D^{1.5+\delta}$, while in our algorithm the dependency of D is $\sim D^{1+\delta}$.

Theorem V.2 (One-shot Expander Pruning)**.** *There is an algorithm $\mathcal{A}$ that can do the following:*

- *$\mathcal{A}$ is given G,D, α_b, δ as inputs: $G = (V, E)$ is an n-node m-edge graph with maximum degree Δ, α_b is a conductance parameter, $\delta \in (0, 1)$ is a parameter, and D is a set of edges where $D \cap E = \emptyset$ where $|D| = O(\alpha_b^2 m/\Delta)$. Let $G_b = (V, E \cup D)$.*
- *Then, in time $\bar{t} = \tilde{O}(\frac{\Delta|D|^{1+\delta}}{\delta\alpha_b^{6+\delta}})$, $\mathcal{A}$ either reports $\phi(G_b) < \alpha_b$, or outputs a* pruning set *$P \subset V$. Moreover, if $\phi(G_b) \geq \alpha_b$, then we have*
  - *$vol_G(P) \leq 2|D|/\alpha_b$, and*
  - *a pruned graph $H = G[V - P]$ has high conductance: $\phi(H) \geq \alpha = \Omega({\alpha_b}^{2/\delta})$.*

We call $\bar{t}$ the *time limit* and α the *conductance guarantee* of $\mathcal{A}$. If we do not care about the time limit, then there is the following algorithm gives a very good conductance guarantee: just find the cut C^* of conductance at most $\alpha_b/10$ that have maximum volume and output $P = C^*$. If $vol(P) > 2|D|/\alpha_b$, then report $\phi(G_b) < \alpha_b$. Otherwise, we must have $\phi(G[V - P]) = \Omega(\alpha_b)$. This can be shown using the result by Spielman and Teng [34, Lemma 7.2]. However, computing the optimum cut C^* is NP-hard.

In [3], they implicitly showed that using only the LBS cut algorithm, which is basically an algorithm for finding a cut similar to C^* but the guarantee is only *approximately* and *locally*, one can quickly obtain the one-shot expander pruning algorithm whose conductance guarantee is not too bad. Below, we explicitly state the reduction in [3]. See the full version of the paper for the proof.

Lemma V.3 ([3])**.** *Suppose there is a $(c_{size}(\sigma), c_{con}(\sigma))$-approximate LBS cut algorithm with running time $t_{LSB}(n, vol(A), \alpha, \sigma)$ when given (G, A, σ, α) as inputs where $G = (V, E)$ is an n-node graph, $A \subset V$ is a set of nodes, σ is an overlapping parameter, and α is a conductance parameter. Then, there is a one-shot expander pruning algorithm with input (G, D, α_b, δ) that has* time limit

$$\bar{t} = O((\frac{|D|}{\alpha_b})^{\delta} \cdot \frac{c_{size}(\alpha_b/2)}{\delta} \cdot t_{LSB}(n, \frac{\Delta|D|}{\alpha_b}, \alpha_b, \alpha_b))$$

and conductance guarantee

$$\alpha = \frac{\alpha_b}{5c_{con}(\alpha_b/2)^{1/\delta - 1}}.$$

Having the above lemma and our new LBS cut algorithm from Section IV, we have that Theorem V.2 follows by setting the right parameters and some simple calculation.

B. Dynamic Expander Pruning

In this section, we exploit the one-shot expander pruning algorithm from Section V-A. To prove Theorem V.1, it is more convenience to prove the more general statement as follows:

Lemma V.4. *There is an algorithm $\mathcal{A}$ that can do the following:*

- *$\mathcal{A}$ is given G_0, α_0, ℓ as inputs: $G_0 = (V, E)$ is an n-node m-edge graph with maximum degree Δ, and $\alpha_0 = \frac{1}{n^\epsilon}$ and ℓ are parameters. Let $P_0 = \emptyset$.*
- *Then G_0 undergoes the sequence of edge deletions of length $T = O(\alpha_0^2 m/\Delta)$.*
- *Given the τ-th update, $\mathcal{A}$ takes $\tilde{O}(\ell^2 \Delta n^{O(1/\ell+\epsilon\ell^\ell)})$ time. Then, $\mathcal{A}$ either reports $\phi(G_0) < \alpha_0$ and halt, or $\mathcal{A}$ updates the pruning set P to P_τ where $P_{\tau-1} \subseteq P_\tau \subseteq V$.*
- *If $\phi(G_0) \geq \alpha_0$ then, for all τ, there exists $W_\tau \subseteq P_\tau$ where $G_\tau[V - W_\tau]$ is connected.*

From Lemma V.4, we immediately obtain Theorem V.1 by choosing the right parameters.

Proof of Theorem V.1: We set $\ell = \frac{\log \frac{1}{\epsilon}}{2 \log \log \frac{1}{\epsilon}}$, so that $\ell^\ell = O(\frac{1}{\epsilon^{1/2}})$. Hence,

$$\begin{aligned} n^{O(1/\ell+\epsilon\ell^\ell)} &= n^{O(\log\log\frac{1}{\epsilon}/\log\frac{1}{\epsilon}+\epsilon^{1/2})} \\ &= n^{O(\log\log\frac{1}{\epsilon}/\log\frac{1}{\epsilon})} = n^{o(1)} \end{aligned}$$

when $\epsilon = o(1)$. We apply Lemma V.4 with this parameters ℓ and $\alpha_0 = \frac{1}{n^\epsilon}$ and we are done. ■

It remains to prove Lemma V.4. See the full version of the paper for the proof. The main idea is to "schedule" many instance of the one-shot pruning algorithm in a clever way.

VI. Pruning on Arbitrary Graphs

In Theorem V.1, we show a fast deterministic algorithm that guarantees connectivity of the pruned graph $G[V - W]$ only when an initial graph is an expander. If the initial graph is not an expander, then there is no guarantee at all. With a simple modification, in this section, we will show a fast randomized algorithm for an arbitrary initial graph that either outputs the desired pruning set or reports failure. Moreover, if the the initial graph is an expander, then it never fails with high probability.

This section is needed in order to make our final algorithm Las Vegas. If we only want a Monte Carlo algorithm, then it is enough to use Theorem V.1 when we combine every component together in Section IX.

Theorem VI.1. *Consider any $\epsilon(n) = o(1)$, and let $\alpha_0(n) = 1/n^{\epsilon(n)}$. There is a dynamic algorithm $\mathcal{A}$ that can maintain a set of nodes P for a graph G undergoing $T = O(m\alpha_0^2(n))$ edge deletions as follows. Let G_τ and P_τ be the graph G and set P after the τ^{th} deletion, respectively.*

- *Initially, in $\tilde{O}(n \log \frac{1}{p})$ time $\mathcal{A}$ sets $P_0 = \emptyset$ and takes as input an n-node m-edge graph $G_0 = (V, E)$ with maximum degree 3.*
- *After the τ^{th} deletion, $\mathcal{A}$ takes $O(n^{O(\log\log\frac{1}{\epsilon(n)}/\log\frac{1}{\epsilon(n)})} \log \frac{1}{p}) = O(n^{o(1)} \log \frac{1}{p})$ time to either 1) report nodes to be added to $P_{\tau-1}$ to form P_τ where*

$$\exists W_\tau \subseteq P_\tau \text{ s.t. } G_\tau[V - W_\tau] \text{ is connected}$$

or 2) reports failure. If $\phi(G_0) \geq \alpha_0(n)$, then $\mathcal{A}$ never fails with probability $1 - p$.

See the full paper for the proof of Theorem VI.1. The key idea is to use the Monte Carlo dynamic spanning forest by Kapron et al. [25] as a certificate for connectivity.

VII. Reduction from Graphs with Few Non-tree Edges Undergoing Batch Insertions

In this section, we show the following crucial reduction:

Theorem VII.1. *Suppose there is a decremental MSF algorithm $\mathcal{A}$ for any m'-edge graph with max degree 3 undergoing a sequence of edge deletions of length $T(m')$, and $\mathcal{A}$ has $t_{pre}(m', p)$ preprocessing time and $t_u(m', p)$ worst-case update time with probability $1 - p$.*

Then, for any numbers B and k where $15k \leq m'$, there is a fully dynamic MSF algorithm $\mathcal{B}$ for any m-edge graph with at most k non-tree edges such that $\mathcal{B}$ can:

- *preprocess the input graph in time*

$$t'_{pre}(m, k, B, p) = t_{pre}(15k, p') + O(m \log^2 m),$$

- *handle a batch of B edge insertions or a single edge deletion in time $t'_u(m, k, B, p)$ which is*

$$O(\frac{B \log k}{k} \cdot t_{pre}(15k, p') + B \log^2 m + \frac{k \log k}{T(k)} + \log k \cdot t_u(15k, p')),$$

where $p' = \Theta(p/\log k)$ and the time guarantee for each operation holds with probability $1 - p$.

The proof of Theorem VII.1 is by extending the reduction by Wulff-Nilsen [2] in two ways. First, the resulting algorithm is more efficient when there are few non-tree edges. Second, the resulting algorithm can also quickly handle a batch of edge insertions.

Although, the extension of the reduction is straightforward and also uses the same "contraction" technique by Henzinger and King [4] and Holm et al. [5], we emphasize that our purpose for using the "contraction" technique is conceptually very different from all previous applications of the (similar) technique [12], [5], [17], [2]. The purpose of all previous applications is for reducing decremental algorithms to fully dynamic algorithms. However, this goal is not crucial for us. Indeed, in our application, by slightly changing the algorithm, the input dynamic MSF algorithm for Theorem VII.1 can also be fully-dynamic and not decremental. But it is very important that the reduction must give an algorithm that is faster when there are few non-tree edges and can handle batch insertions. Therefore, this work illustrates a new application of the "contraction" technique.

There are previous attempts for speeding up the algorithm when there are few non-tree edges. In the dynamic SF algorithm of Nanongkai and Saranurak [3] and the dynamic MSF algorithm of Wulff-Nilsen [2], they both also devised the algorithms that run on a graph with k non-tree edges by extending the 2-dimensional topology tree of

Frederickson [6]. The algorithms have $O(\sqrt{k})$ update time. In the context of [3], [2], they have $k = n^{1-\epsilon_0}$ for some small constant $\epsilon_0 > 0$ where n is the number of nodes, and hence $O(\sqrt{k}) = O(n^{0.5-\epsilon_0/2})$. This eventually leads to their dynamic SF and MSF algorithms with update time $n^{0.5-\Omega(1)}$.

In our application paper, we will have $k = n^{1-o(1)}$ and the update time of $O(\sqrt{k})$ is too slow. Fortunately, using the reduction from this section, we can reduce to the problem where the algorithm runs on graphs with only $O(k)$ edges, and then recursively run our algorithm on that graph. Together with other components, this finally leads to the algorithm with subpolynomial update time.

See the full version of the paper for the proof of Theorem VII.1.

VIII. MSF Decomposition

In this section, we show an improved algorithm for computing a hierarchical decomposition of a graph called *MSF decomposition*. This decomposition is introduced by Wulff-Nilsen [2, Section 3.1] and it is the main subroutine in the preprocessing algorithm of his dynamic MSF algorithm and also of ours. Our improved algorithm has a better trade-off between the running time and the "quality" of the decomposition as will be made precise later. The improved version is obtained simply by using the flow-based *expansion decomposition* algorithm[11] by Nanongkai and Saranurak [3] as the main subroutine, instead of using diffusion/spectral-based algorithms as in [2]. Moreover, as the expansion decomposition algorithm is defined based on *expansion* and not conductance, this is easier to work with and it simplifies some steps of the algorithm.Before stating the main result in Theorem VIII.3, we need the following definition:

Definition VIII.1 (Hierarchical Decomposition)**.** *For any graph $G = (V, E)$, a* hierarchical decomposition $\mathcal{H}$ *of G is a rooted tree. Each node $C \in \mathcal{H}$ corresponds to some subgraph of G which is called a* cluster. *There are two conditions that $\mathcal{H}$ needs to satisfy: 1) the root cluster of $\mathcal{H}$ corresponds to the graph G itself, 2) for each non-leaf cluster $C \in \mathcal{H}$, let $\{C'_i\}_i$ be the children of C. Then vertices of $\{C'_i\}_i$ form a partition of vertices in C, i.e. $V(C) = \dot{\bigcup}_i V(C'_i)$. The root cluster is a* level-1 *cluster. A child of* level-i cluster *is a* level-$(i+1)$ cluster. *The* depth *of $\mathcal{H}$ is the depth of the tree. Let $E^C = E(C) - \dot{\bigcup}_i E(C'_i)$ be the set of edges in C which are not edges in any of C'_i's. We call an edge $e \in E^C$ a C-*own *edge, and an edge $f \in E(C) - E^C = \dot{\bigcup}_i E(C'_i)$ a C-*child *edge.*

We note that, for any cluster C with a child C', it is possible that $E(C')$ is a proper subset of $E(C[V(C')])$. That is, there might be some edge $e = (u, v) \in E(C)$ where $u, v \in V(C')$ but $e \notin E(C')$. In other words, there can be a C-own edge (u, v) where both $u, v \in V(C')$. Observe the following:

Fact VIII.2. *Let $\mathcal{H}$ be a hierarchical decomposition of a graph $G = (V, E)$. Then $\dot{\bigcup}_{C \in \mathcal{H}} E^C = E$.*

Throughout this section, we assume that, in an input graph with m-edge, the edges have distinct weights ranging from number 1 to m. Throughout this section, let $\gamma = n^{O(\sqrt{\log \log n / \log n})} = n^{o(1)}$ where n is the number of nodes in a graph. The main result of this section is the below theorem:

Theorem VIII.3. *There is a randomized algorithm called* MSF decomposition*, MSFdecomp, which takes the following as input:*

- *a connected graph $G = (V, E, w)$ with n nodes, m edges and max degree 3, where $w : E \to \{1, \ldots, m\}$ is the weight function of edges in G,*
- *a failure probability parameter $p \in (0, 1]$, a conductance parameter $\alpha \in [0, 1]$, and parameters $d \geq 3$, s_{low} and s_{high} where $s_{high} \geq s_{low}$.*

In time $\tilde{O}(nd\gamma \log \frac{1}{p})$ where $\gamma = n^{O(\sqrt{\log \log n / \log n})}$, the algorithm returns (i) a graph $G' = (V, E, w')$ with a new weight function $w' : E \to \mathbb{R}$ and (ii) a hierarchical decomposition $\mathcal{H}$ of the re-weighted graph G' with following properties:

1) *For all $e \in E$, $w'(e) \geq w(e)$.*
2) *$|\{e \in E \mid w(e) \neq w'(e)\}| \leq \alpha d\gamma n$.*
3) *For any cluster $C \in \mathcal{H}$ and any set of edges D, $\mathsf{MSF}(C - D) = \dot{\bigcup}_{C'\text{:child of } C} \mathsf{MSF}(C' - D) \dot{\cup} (\mathsf{MSF}(C - D) \cap (E^C - D))$.*
4) *$\mathcal{H}$ has depth at most d.*
5) *A cluster C is a leaf cluster iff $E(C) \leq s_{high}$.*
6) *Each leaf cluster contains at least $s_{low}/3$ nodes.*
7) *For level i, $|\dot{\bigcup}_{C\text{:non-leaf, level-}i} E^C| \leq n/(d-2) + \alpha\gamma n$.*
8) *With probability $1 - p$, all non-root clusters $C \in \mathcal{H}$ are such that $\phi(C) = \Omega(\alpha / s_{low})$.*

We call the lower bound of conductance for all non-root clusters is the *conductance guarantee* of the hierarchical decomposition $\mathcal{H}$, which is $\Omega(\alpha/s_{low})$ in our algorithm. Compared with the MSF decomposition algorithm in [2, Section 3.1], our algorithm runs significantly faster and has a better trade-off guarantee between conductance of the cluster and the number of edges re-weighted. In particular, the running time of our algorithm does not depends on the conductance parameter α.

Now, we give some intuition why this decomposition can be useful in our application. Given an input n-node graph G, we set $\alpha = 1/\gamma^3$, $d = \gamma$, $s_{low} = \gamma$, and $s_{high} = n/\gamma$. The algorithm increases the weight of only $(1/\gamma)$-fraction of edges resulting in the re-weighted graph G', and then it outputs the hierarchy decomposition $\mathcal{H}$ of G'.

[11]The expansion decomposition algorithm was used as a main preprocessing algorithm for their dynamic SF algorithm.

Property 3 of $\mathcal{H}$ is crucial and it implies that $\mathsf{MSF}(G') = \dot{\bigcup}_{C\in\mathcal{H}}(\mathsf{MSF}(C) \cap E^C)$, and this holds even after deleting any set of edges. This suggests that, to find $\mathsf{MSF}(G)$, we just need separately find $\mathsf{MSF}(C) \cap E^C$, i.e., the C-own edges that are in $\mathsf{MSF}(C)$, for every cluster $C \in \mathcal{H}$. That is, the task of maintaining the MSF is also "decomposed" according the decomposition. Other properties are about bounding the size of some sets of edges and the conductance of clusters. These properties will allow our dynamic MSF algorithm to have fast update time.

See the full version of the paper for the proof of Theorem VIII.3.

IX. Dynamic MSF Algorithm

In this section, we prove the main theorem:

Theorem IX.1. *There is a fully dynamic MSF algorithm on an n-node m-edge graph that has preprocessing time $O(m^{1+O(\sqrt{\log\log m/\log m})}\log\frac{1}{p}) = O(m^{1+o(1)}\log\frac{1}{p})$ and worst-case update time $O(n^{O(\log\log\log n/\log\log n)}\log\frac{1}{p}) = O(n^{o(1)}\log\frac{1}{p})$ with probability $1-p$.*

By using a standard reduction or a more powerful reduction from Theorem VII.1, it is enough to show the following:

Lemma IX.2. *There is a decremental MSF algorithm $\mathcal{A}$ on an n-node m-edge graph G with max degree 3 undergoing a sequence of edge deletions of length $T = \Theta(n^{1-O(\log\log\log n/\log\log n)})$. $\mathcal{A}$ has preprocessing time $O(n^{1+O(\sqrt{\log\log n/\log n})}\log\frac{1}{p})$ and worst-case update time $O(n^{O(\log\log\log n/\log\log n)}\log\frac{1}{p})$ with probability $1-p$.*

We note that essentially all the ideas in this section, in particular the crucial definition of *compressed clusters*, already appeared in Wulff-Nilsen [2]. In this section, we only make sure that, with our improved tools from previous sections, we can integrate all of them using the same approach as in [2]. Obviously, the run time analysis must change because our algorithm is faster and need somewhat more careful analysis. Although the correctness follows as in [2], the terminology changes a bit because MSF decomposition from Theorem VIII.3 is presented in a more modular way.

The high-level idea in [2] of the algorithm $\mathcal{A}$ is simple. To maintain $\mathsf{MSF}(G)$, we maintain a graph H, called the *sketch graph*, where at any time $\mathsf{MSF}(G) = \mathsf{MSF}(H)$ and H contains only few non-tree edges with high probability. Then we just maintain $\mathsf{MSF}(H)$ using another algorithm for graphs with few non-tree edges. See the full version of the paper for the proof of Lemma IX.2.

X. Open Problems

Dynamic MSF.

First, it is truly intriguing whether there is a *deterministic* algorithm that is as fast as our algorithm. The current best update time of deterministic algorithms is still $\tilde{O}(\sqrt{n})$ [6], [8], [27] (even for dynamic connectivity). Improving this bound to $O(n^{0.5-\Omega(1)})$ will already be a major result. Secondly, can one improve the $O(n^{o(1)})$ update time to $O(\text{polylog}(n))$? There are now several barriers in our approach and this improvement should require new ideas. Lastly, it is also very interesting to simplify our algorithm.

Expander-related Techniques.

The combination of the expansion decomposition and dynamic expander pruning might be useful for other dynamic graph problems. Problems whose static algorithms are based on low-diameter decomposition (e.g. low-stretch spanning tree) are possible candidates. Indeed, it is conceivable that the expansion decomposition together with dynamic expander pruning can be used to maintain low diameter decomposition under edge updates, but additional work maybe required.

Worst-case Update Time Against Adaptive Adversaries.

Among major goals for dynamic graph algorithm are (1) to reduce gaps between *worst-case* and *amortized* update time, and (2) to reduce gaps between update time of algorithms that work against *adaptive* adversaries and those that require *oblivious* adversaries. Upper bounds known for the former case (for both goals) are usually much higher than those for the latter. However, worst-case bounds are crucial in real-time applications, and being against adversaries is often needed when algorithms are used as subroutines of static algorithms. Note that of course deterministic algorithms always work against adaptive adversaries.

The result in this paper is a step towards both goals. The best amortized bound for dynamic MSF is $O(\text{polylog}(n))$ [5], [17]. For dynamic SF problem, the result by [25], [26] implies the current best algorithm against oblivious adversaries with $O(\text{polylog}(n))$ worst-case update time. Our dynamic MSF algorithm is against adaptive adversaries and has $O(n^{o(1)})$ worst-case update time. This significantly reduces the gaps on both cases.

It is a challenging goal to do the same for other fundamental problems. For example, dynamic 2-edge connectivity has $O(\text{polylog}(n))$ amortized update time [5] but only $O(\sqrt{n})$ worst-case bound [35], [8]. Dynamic APSP has $\tilde{O}(n^2)$ amortized bound [36] but only $\tilde{O}(n^{2+2/3})$ worst-case bound [22]. There are fast algorithms against oblivious adversaries for dynamic maximal matching [37], spanner [38], and cut/spectral sparsifier [10]. It will be exciting to have algorithms against adaptive adversaries with comparable update time for these problems.

XI. 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. Nanongkai and Saranurak were also

partially supported by the Swedish Research Council (Reg. No. 2015-04659).

REFERENCES

[1] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen, "Dynamic minimum spanning forest with subpolynomial worst-case update time," *CoRR*, vol. abs/1708.03962, 2017. [Online]. Available: https://arxiv.org/abs/1708.03962

[2] C. 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*, 2017, pp. 1130–1143. [Online]. Available: http://doi.acm.org/10.1145/3055399.3055415

[3] D. Nanongkai and T. 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*, 2017, pp. 1122–1129. [Online]. Available: http://doi.acm.org/10.1145/3055399.3055447

[4] M. R. Henzinger and V. King, "Fully dynamic 2-edge-connectivity algorithm in polylogarithmic time per operation," Digital Equipment Corp., Systems Research Ctr., 130 Lytton Rd., Palo Alto, CA, 94301, USA, Technical note 1997-004, 12 Jun 1997.

[5] J. Holm, K. de Lichtenberg, and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *J. ACM*, vol. 48, no. 4, pp. 723–760, 2001, announced at STOC 1998. [Online]. Available: http://doi.acm.org/10.1145/502090.502095

[6] G. N. Frederickson, "Data structures for on-line updating of minimum spanning trees, with applications," *SIAM J. Comput.*, vol. 14, no. 4, pp. 781–798, 1985, announced at STOC'83. [Online]. Available: http://dx.doi.org/10.1137/0214055

[7] M. Thorup and D. R. Karger, "Dynamic graph algorithms with applications," in *Algorithm Theory-SWAT 2000*. Springer, 2000, pp. 1–9.

[8] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, "Sparsification - a technique for speeding up dynamic graph algorithms," *J. ACM*, vol. 44, no. 5, pp. 669–696, 1997, announced at FOCS 1992. [Online]. Available: http://doi.acm.org/10.1145/265910.265914

[9] M. Thorup, "Fully-dynamic min-cut," *Combinatorica*, vol. 27, no. 1, pp. 91–127, 2007, announced at STOC'01. [Online]. Available: http://dx.doi.org/10.1007/s00493-007-0045-2

[10] I. Abraham, D. Durfee, I. Koutis, S. Krinninger, and R. Peng, "On fully dynamic graph sparsifiers," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, 2016, pp. 335–344. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2016.44

[11] M. R. Henzinger and V. King, "Randomized fully dynamic graph algorithms with polylogarithmic time per operation," *J. ACM*, vol. 46, no. 4, pp. 502–516, 1999, announced at STOC 1995. [Online]. Available: http://doi.acm.org/10.1145/320211.320215

[12] ——, "Maintaining minimum spanning trees in dynamic graphs," in *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, 1997, pp. 594–604. [Online]. Available: http://dx.doi.org/10.1007/3-540-63165-8_214

[13] M. R. Henzinger and M. Thorup, "Sampling to provide or to bound: With applications to fully dynamic graph algorithms," *Random Struct. Algorithms*, vol. 11, no. 4, pp. 369–379, 1997. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1098-2418(199712)11:4⟨369::AID-RSA5⟩3.0.CO;2-X

[14] M. Thorup, "Near-optimal fully-dynamic graph connectivity," in *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, F. F. Yao and E. M. Luks, Eds. ACM, 2000, pp. 343–350. [Online]. Available: http://doi.acm.org/10.1145/335305.335345

[15] S.-E. Huang, D. Huang, T. Kopelowitz, and S. Pettie, "Fully dynamic connectivity in $o(\log n(\log \log n)^2)$ amortized expected time," in *SODA*, 2017.

[16] C. Wulff-Nilsen, "Faster deterministic fully-dynamic graph connectivity," in *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, 2013, pp. 1757–1769. [Online]. Available: http://dx.doi.org/10.1137/1.9781611973105.126

[17] J. Holm, E. Rotenberg, and C. Wulff-Nilsen, "Faster fully-dynamic minimum spanning forest," in *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, 2015, pp. 742–753. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-48350-3_62

[18] M. Patrascu and E. D. Demaine, "Logarithmic lower bounds in the cell-probe model," *SIAM J. Comput.*, vol. 35, no. 4, pp. 932–963, 2006, announced at SODA'04 and STOC'04. [Online]. Available: http://dx.doi.org/10.1137/S0097539705447256

[19] M. Patrascu and M. 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*, 2007, pp. 263–271. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/FOCS.2007.54

[20] P. Sankowski, "Dynamic transitive closure via dynamic matrix inverse," in *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, 2004, pp. 509–517. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/FOCS.2004.25

[21] M. Thorup, "Worst-case update times for fully-dynamic all-pairs shortest paths," in *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, H. N. Gabow and R. Fagin, Eds. ACM, 2005, pp. 112–119. [Online]. Available: http://doi.acm.org/10.1145/1060590.1060607

[22] I. Abraham, S. Chechik, and S. Krinninger, "Fully dynamic all-pairs shortest paths with worst-case update-time revisited," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, 2017, pp. 440–452. [Online]. Available: http://dx.doi.org/10.1137/1.9781611974782.28

[23] G. Bodwin and S. Krinninger, "Fully dynamic spanners with worst-case update time," in *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, 2016, pp. 17:1–17:18. [Online]. Available:

https://doi.org/10.4230/LIPIcs.ESA.2016.17

[24] S. Bhattacharya, M. Henzinger, and D. Nanongkai, "Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, 2017, pp. 470–489. [Online]. Available: http://dx.doi.org/10.1137/1.9781611974782.30

[25] B. M. Kapron, V. King, and B. Mountjoy, "Dynamic graph connectivity in polylogarithmic worst case time," in *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, S. Khanna, Ed. SIAM, 2013, pp. 1131–1142. [Online]. Available: http://dx.doi.org/10.1137/1.9781611973105.81

[26] D. Gibb, B. M. Kapron, V. King, and N. Thorn, "Dynamic graph connectivity with improved worst case update time and sublinear space," *CoRR*, vol. abs/1509.06464, 2015. [Online]. Available: http://arxiv.org/abs/1509.06464

[27] C. Kejlberg-Rasmussen, T. Kopelowitz, S. Pettie, and M. Thorup, "Faster worst case deterministic dynamic connectivity," in *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, ser. LIPIcs, P. Sankowski and C. D. Zaroliagis, Eds., vol. 57. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 53:1–53:15. [Online]. Available: http://dx.doi.org/10.4230/LIPIcs.ESA.2016.53

[28] R. Peng, "Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time," in *SODA*. SIAM, 2016, pp. 1862–1867.

[29] L. Orecchia and Z. 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*, 2014, pp. 1267–1286. [Online]. Available: http://dx.doi.org/10.1137/1.9781611973402.94

[30] M. Henzinger, S. Rao, and D. Wang, "Local flow partitioning for faster edge connectivity," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, 2017, pp. 1919–1938. [Online]. Available: http://dx.doi.org/10.1137/1.9781611974782.125

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 6.

[32] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Comput. Syst. Sci.*, vol. 26, no. 3, pp. 362–391, 1983. [Online]. Available: http://dx.doi.org/10.1016/0022-0000(83)90006-5

[33] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup, "Maintaining information in fully dynamic trees with top trees," *ACM Transactions on Algorithms*, vol. 1, no. 2, pp. 243–264, 2005. [Online]. Available: http://doi.acm.org/10.1145/1103963.1103966

[34] D. A. Spielman and S. Teng, "Spectral sparsification of graphs," *SIAM J. Comput.*, vol. 40, no. 4, pp. 981–1025, 2011.

[35] G. N. Frederickson, "Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees," *SIAM J. Comput.*, vol. 26, no. 2, pp. 484–538, 1997, announced at FOCS 1991. [Online]. Available: http://dx.doi.org/10.1137/S0097539792226825

[36] C. Demetrescu and G. F. Italiano, "A new approach to dynamic all pairs shortest paths," in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, 2003, pp. 159–166. [Online]. Available: http://doi.acm.org/10.1145/780542.780567

[37] S. Baswana, M. Gupta, and S. 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*, 2011, pp. 383–392. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2011.89

[38] S. Baswana, S. Khurana, and S. Sarkar, "Fully dynamic randomized algorithms for graph spanners," *ACM Trans. Algorithms*, vol. 8, no. 4, p. 35, 2012. [Online]. Available: http://doi.acm.org/10.1145/2344422.2344425