A Deterministic Algorithm for Balanced Cut with Applications to Dynamic Connectivity, Flows, and Beyond

Julia Chuzhoy, *TTIC*, *USA*
Yu Gao, *Georgia Tech*, *USA*
Jason Li, *CMU*, *USA*
Danupon Nanongkai, *KTH*, *Sweden*
Richard Peng, *Georgia Tech*, *USA*
Thatchaphol Saranurak, *TTIC*, *USA*

***Abstract*—We consider the classical Minimum Balanced Cut problem: given a graph G, compute a partition of its vertices into two subsets of roughly equal volume, while minimizing the number of edges connecting the subsets. We present the first *deterministic, almost-linear time* approximation algorithm for this problem. Specifically, our algorithm, given an n-vertex m-edge graph G and any parameter $1 \leq r \leq O(\log n)$, computes a $(\log m)^{r^2}$-approximation for Minimum Balanced Cut in G, in time $O\left(m^{1+O(1/r)+o(1)} \cdot (\log m)^{O(r^2)}\right)$. In particular, we obtain a $(\log m)^{1/\epsilon}$-approximation in time $m^{1+O(\sqrt{\epsilon})}$ for any constant $\epsilon > 0$, and a $(\log m)^{f(m)}$-approximation in time $m^{1+o(1)}$, for any slowly growing function $f(m)$. We obtain deterministic algorithms with similar guarantees for the Sparsest Cut and the Lowest-Conductance Cut problems.**

Our algorithm for the Minimum Balanced Cut problem in fact provides a stronger guarantee: it either returns a balanced cut whose value is close to a given target value, or it certifies that such a cut does not exist by exhibiting a large subgraph of G that has high conductance. We use this algorithm to obtain deterministic algorithms for dynamic connectivity and minimum spanning forest, whose worst-case update time on an n-vertex graph is $n^{o(1)}$, thus resolving a major open problem in the area of dynamic graph algorithms. Our work also implies deterministic algorithms for a host of additional problems, whose time complexities match, up to subpolynomial in n factors, those of known randomized algorithms. The implications include almost-linear time deterministic algorithms for solving Laplacian systems and for approximating maximum flows in undirected graphs.

***Keywords*-deterministic algorithms; dynamic connectivity; balanced cuts; maximum flow; Laplacian solvers;**

A full version of the paper is available at https://arxiv.org/abs/1910.08025.

I. Introduction

In the classical Minimum Balanced Cut problem, the input is an n-vertex graph $G = (V, E)$, and the goal is to compute a partition of V into two subsets A and B with $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/3$, while minimizing $|E_G(A, B)|$; here, $E_G(A, B)$ is the set of all edges with one endpoint in A and another in B, and, for a set S of vertices of G, $\mathrm{Vol}_G(S)$ denotes its *volume* – the sum of the degrees of all vertices of S. Lastly, $\mathrm{Vol}(G) = \mathrm{Vol}_G(V)$ is the total volume of the graph. The Minimum Balanced Cut problem is closely related to the Minimum-Conductance Cut problem, where the goal is to compute a subset S of vertices of minimum *conductance*, defined as $|E_G(S, V \setminus S)|/\min\{\mathrm{Vol}_G(S), \mathrm{Vol}_G(V \setminus S)\}$, and to the Sparsest Cut problem, where the goal is to compute a subset S of vertices of minimum *sparsity*: $|E_G(S, V \setminus S)|/\min\{|S|, |V \setminus S|\}$. While all three problems are known to be NP-hard, approximation algorithms for them are among the most central and widely used tools in algorithm design, especially due to their natural connections to the hierarchical divide-and-conquer paradigm [1], [2], [3], [4], [5], [6], [7]. We note that approximation algorithms for Minimum Balanced Cut often consider a relaxed (or a bi-criteria) version, where we only require that the solution (A, B) returned by the algorithm satisfies $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/4$, but the solution value is compared to that of the optimal balanced cut.

The first approximation algorithm for Minimum Balanced Cut, whose running time is near-linear in the graph size, was developed in the seminal work of Spielman and Teng [2]. This algorithm was used in [2] in order to decompose a given graph into a collection of "near-expanders", which are then exploited in order to construct spectral sparsifiers, eventually leading to an algorithm for solving systems of linear equations in near-linear time. Algorithms for Minimum Balanced Cut also served as crucial building blocks in the more recent breakthrough results that designed near- and almost-linear time[1] approximation algorithms for a large class of flow and regression problems [8], [9], [10], [11] and faster exact algorithms for maximum flow, shortest paths with negative weights, and minimum-cost flow [12], [13]. Spielman and Teng's expander decomposition was later strengthened by Nanongkai, Saranurak and Wulff-Nilsen [7], [14], [15], who used it to obtain algorithms for the dynamic minimum spanning forest problem with improved worst-case update time. The fastest current algorithm for computing expander decompositions is due to Saranurak and Wang [16]; a similar decomposition was recently used by Chuzhoy and Khanna [17] in their algorithm for the decremental

[1]We informally say that an algorithm runs in near-linear time, if its running time is $O(m \cdot \mathrm{poly}\log n)$, where m and n are the number of edges and vertices in the input graph, respectively. We say that the running time is almost-linear, if it is bounded by $m^{1+o(1)}$.

2575-8454/20/$31.00 ©2020 IEEE
DOI 10.1109/FOCS46700.2020.00111

single-source shortest paths problem, that in turn led to a faster algorithm for approximate vertex-capacitated maximum flow.

Unfortunately, all algorithms mentioned above are *randomized*. This is mainly because all existing almost- and near-linear time algorithms for Minimum Balanced Cut are randomized [2], [18]. A fundamental open question in this area is then: is there a *deterministic* algorithm for Minimum Balanced Cut with similar performance guarantees? Resolving this questions seems a key step to obtaining fast deterministic algorithms for all aforementioned problems, and to resolving one of the most prominent open problems in the area of dynamic graph algorithms, namely, whether there is a deterministic algorithm for Dynamic Connectivity, whose worst-case update time is smaller than the classical $O(\sqrt{n})$ bound of Frederickson [19], [20] by a factor that is polynomial in n.

The best previous published bound on the running time of a determinsitic algorithm for Minimum Balanced Cut is $O(mn)$ [21]. A recent manuscript by a subset of the authors, together with Yingchareonthawornchai [22], obtains a running time of $\min\left\{n^{\omega+o(1)}, m^{1.5+o(1)}\right\}$, where $\omega < 2.372$ is the matrix multiplication exponent, and n and m are the number of nodes and edges of the input graph, respectively. This algorithm is used in [22] to obtain faster deterministic algorithms for the vertex connectivity problem. However, the running time of the algorithm of [22] for Minimum Balanced Cut is somewhat slow, and it just falls short of breaking the $O(\sqrt{n})$ worst-case update time bound for Dynamic Connectivity.

A. Our Results

We present a deterministic (bi-criteria) algorithm for Minimum Balanced Cut that, for any parameter $r = O(\log n)$, achieves an approximation factor $\alpha(r) = (\log m)^{r^2}$ in time $T(r) = O\left(m^{1+O(1/r)+o(1)} \cdot (\log m)^{O(r^2)}\right)$, where n and m are the number of vertices and edges in the input graph, respectively. In particular, for any constant $\epsilon > 0$, the algorithm achieves $(\log m)^{1/\epsilon}$-approximation in time $O\left(m^{1+O(\sqrt{\epsilon})}\right)$. For any slowly growing function $f(m)$ (for example, $f(m) = \log\log m$ or $f(m) = \log^* m$), it achieves $(\log m)^{f(m)}$-approximation in time $m^{1+o(1)}$.

In fact our algorithm provides somewhat stronger guarantees: it either computes an almost balanced cut whose value is within an $\alpha(r)$ factor of a given target value η; or it certifies that every balanced cut in G has value $\Omega(\eta)$, by producing a large sub-graph of G that has a large conductance. This algorithm implies fast deterministic algorithms for all the above mentioned problems, including, in particular, improved worst-case update time guarantees for (undirected) Dynamic Connectivity and Minimum Spanning Forest.

In order to provide more details on our results and techniques, we need to introduce some notation. Throughout, we assume that we are given an m-edge, n-node undirected graph, denoted by $G = (V, E)$. A *cut* in G is a partition (A, B) of V into two non-empty subsets; abusing the notation, we will also refer to subsets S of vertices with $S \neq \emptyset, V$ as cuts, meaning the partition $(S, V \setminus S)$ of V. The *conductance* of a cut S in G, that was already mentioned above, is defined as:

$$\Phi_G(S) := \frac{|E_G(S, V \setminus S)|}{\min\{\mathrm{Vol}_G(S), \mathrm{Vol}_G(V \setminus S)\}},$$

and the conductance of a graph G, that we denote by $\Phi(G)$, is the smallest conductance of any cut S of G: $\Phi(G) := \min_{S \subsetneq V : S \neq \emptyset}\{\Phi_G(S)\}$.

A notion that is closely related to conductance is that of sparsity. The *sparsity* of a cut S in G is: $\Psi_G(S) := \frac{|E_G(S, V \setminus S)|}{\min\{|S|, |V \setminus S|\}}$, and the *expansion* of the graph G is the minimum sparsity of any cut S in G: $\Psi(G) := \min_{S \subsetneq V : S \neq \emptyset}\{\Psi_G(S)\}$.

We say that a cut S is *balanced* if $\mathrm{Vol}_G(S), \mathrm{Vol}_G(V \setminus S) \geq \mathrm{Vol}(G)/3$. The main tool that we use in our approximation algorithm for the Minimum Balanced Cut problem is the BalCutPrune problem, that is defined next. Informally, the problem seeks to either find a low-conductance balanced cut in a given graph, or to produce a certificate that every balanced cut has a high conductance, by exhibiting a large sub-graph of G that has a high conductance.

Definition 1 (BalCutPrune problem)**.** The input to the α-approximate BalCutPrune problem is a graph $G = (V, E)$, a conductance parameter $0 < \phi \leq 1$, and an approximation factor α. The goal is to compute a partition (A, B) of $V(G)$ (where possibly $B = \emptyset$), with $|E_G(A, B)| \leq \alpha\phi \cdot \mathrm{Vol}(G)$, such that one of the following holds: either

1) **(Cut)** $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/3$; or
2) **(Prune)** $\mathrm{Vol}_G(A) \geq \mathrm{Vol}(G)/2$, and graph $G[A]$ has conductance at least ϕ.

Our main technical result is the following.

Theorem 2 (Main Result)**.** *There is a deterministic algorithm, that, given a graph G with m edges, and parameters $\phi \in (0, 1]$, $1 \leq r \leq O(\log n)$, and $\alpha = (\log m)^{r^2}$, computes a solution to the α-approximate* BalCutPrune *problem instance (G, ϕ) in time $O\left(m^{1+O(1/r)+o(1)} \cdot (\log m)^{O(r^2)}\right)$.*

In particular, by letting r be a large constant, we obtain a $(\log m)^{1/\epsilon}$-approximation in time $m^{1+O(\sqrt{\epsilon})}$ for any constant $\epsilon > 0$, and by letting $f(m)$ be any slowly growing function (for example, $f(m) = \log\log m$ or $f(m) = \log^* m$), and setting $r = \sqrt{f(m)}$, we obtain $(\log m)^{f(m)}$-approximation in time $m^{1+o(1)}$.

The algorithm from Theorem 2 immediately implies a deterministic bi-criteria factor-$(\log n)^{r^2}$-approximation algorithm for Minimum Balanced Cut, with running time $O\left(m^{1+O(1/r)+o(1)} \cdot (\log m)^{O(r^2)}\right)$ for any value of $r \leq$

$O(\log m)$. Indeed, suppose we are given any conductance parameter $0 < \phi < 1$ for an input graph $G = (V, E)$. We apply the algorithm from Theorem 2 to graph G with the parameter ϕ, obtaining a partition (A, B) in $V(G)$, with $|E_G(A, B)| \leq \alpha\phi \cdot \mathrm{Vol}(G)$. If $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/4$, then we obtain an (almost) balanced cut (A, B) of conductance at most $\alpha\phi$. Otherwise, we are guaranteed that $\mathrm{Vol}_G(A) \geq 3\mathrm{Vol}(G)/4$, and graph $G[A]$ has conductance at least ϕ. We claim that in this case, for any balanced cut (A', B') in G, $|E_G(A', B')| \geq \Omega(\phi \cdot \mathrm{Vol}(G))$ holds. This is because any such partition (A', B') of V defines a partition (X, Y) of A, with $\mathrm{Vol}_G(X), \mathrm{Vol}_G(Y) \geq \Omega(\mathrm{Vol}(G))$, and, since $\Phi(G[A]) \geq \phi$, we get that $|E_G(X, Y)| \geq \Omega(\phi \cdot \mathrm{Vol}(G))$. Therefore, we obtain the following corollary.

Corollary 3. *There is an algorithm that, given an n-vertex m-edge graph G, a target value η and a parameter $r \leq O(\log n)$, either returns a partition (A, B) of $V(G)$ with $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/4$ and $|E_G(A, B)| \leq \alpha(r) \cdot \eta$, for $\alpha(r) = (\log m)^{r^2}$, or it certifies that for any partition (A, B) of $V(G)$ with $\mathrm{Vol}_G(A), \mathrm{Vol}_G(B) \geq \mathrm{Vol}(G)/3$, $|E_G(A, B)| \geq \Omega(\eta)$ must hold. The running time of the algorithm is $O\left(m^{1+O(1/r)+o(1)} \cdot (\log m)^{O(r^2)}\right)$.*

Algorithms for Minimum Balanced Cut often differ in the type of certificate that they provide when the value of the Minimum Balanced Cut is greater than the given threshold (that corresponds to the (Prune) case in Definition 1). The original near-linear time algorithm of Spielman and Teng [2] outputs a set S of nodes of small volume, with the guarantee that for some subset $S' \subseteq S$, the graph $G - S'$ has high conductance. This guarantee, however, is not sufficient for several applications. A version that was found to be more useful in several recent applications, such as e.g. Dynamic Connectivity [16], [7], [14], [15], is somewhat similar to that in the definition of BalCutPrune, but with a somewhat stronger guarantee in the (Prune) case[2].

The approximation factor α of Spielman and Teng's algorithm [2] depends on the parameter ϕ, and its time complexity depends on both ϕ and α. Several subsequent papers have improved the approximation factor or the time complexity of their algorithm e.g. [18], [21], [23], [24], [25]; we do not discuss these results here since they are not directly relevant to this work.

B. Applications

An immediate consequence of our results is deterministic algorithms for the Sparsest Cut and the Lowest-Conductance Cut problems, summarized in the next theorem.

[2]To be precise, that version requires that $|E_G(A, B)| \leq \alpha\phi \cdot \mathrm{Vol}_G(B)$, which is somewhat stronger than our requirement that $|E_G(A, B)| \leq \alpha \cdot \phi \cdot \mathrm{Vol}(G)$. But for all applications we consider, our guarantee still suffices, possibly because the two guarantees are essentially the same when the cut (A, B) is balanced.

Theorem 4. *There is a deterministic algorithm, that, given an n-vertex and m-edge graph G, and a parameter $r \leq O(\log n)$, computes a $(\log n)^{r^2}$-approximate solution for the Sparsest Cut problem on G, in time $O\left(m^{1+O(1/r)+o(1)} \cdot (\log n)^{O(r^2)}\right)$. Similarly, there is a deterministic algorithm that achieves similar performance guarantees for the Lowest-Conductance Cut problem.*

We note that the best current deterministic approximation algorithm for both Sparsest Cut and Lowest-Conductance Cut, due to Arora, Rao and Vazirani [26], achieves an $O(\sqrt{\log n})$-approximation. Unfortunately, the algorithm has a large (but polynomially bounded) running time, since it needs to solve an SDP deterministically. There are faster $O(\log n)$-approximation deterministic algorithms for both the Sparsest Cut and the Lowest-Conductance problems, with running time $\tilde{O}(m^2)$, that are based on the Multiplicative Weights Update framework [27], [28]. If we allow the approximation ratio to depend on ϕ, where ϕ is the value of the optimal solution, then there are several algorithms that are based on a spectral approach for both problems. The algorithm of [29] computes a cut with conductance at most $O(\phi^{1/2})$ in time $\tilde{O}(n^{\omega})$. Using the Personalized PageRank algorithm [21], a cut of conductance at most $O(\phi^{1/2})$ can found in time $\tilde{O}(mn)$. Recently, Gao et. al [22] provided an algorithm to compute a cut of conductance at most $\phi^{1/2}n^{o(1)}$, in time $O(m^{1.5+o(1)})$.[3]

Additionally, we obtain faster deterministic algorithms for a number of other cut and flow problems; the performance of our algorithms matches that of the best current randomized algorithms, to within factor $n^{o(1)}$. We summarize these bounds in Table I and Table II. We now turn to discuss the implications of our results to the Dynamic Connectivity problem, which was one of the main motivations of this work.

In the most basic version of the Dynamic Connectivity problem, we are given a graph G that undergoes edge deletions and insertions, and the goal is to maintain the information of whether G is connected. The Dynamic Connectivity problem and its generalizations – dynamic Spanning Forest (SF) and dynamic Minimum Spanning Forest (MSF) – have played a central role in the development of the area of dynamic graph algorithms for over three decades (see, e.g., [15], [7] for further discussions).

An important measure of the performance of a dynamic algorithm is its *update time* – the amount of time that is needed in order to process each update (an insertion or a deletion of an edge). We distinguish between *amortized update time*, that upper-bounds the average time that the algorithm spends on each update, and *worst-case update time*, that upper-bounds the largest amount of time that the algorithm ever spends on a single update.

[3]The last two algorithms, in fact, provide additional guarantees regarding the balance of the returned cut.

Problem	Best previous running time: deterministic	Best previous running time: randomized	Our results: deterministic
$(1+\epsilon)$-approximate undirected max-flow/min-cut	$\tilde{O}(m \min\{m^{1/2}, n^{2/3}\})$ [30] (exact)	$\tilde{O}(m\epsilon^{-1})$ [31], [9], [25]	$\widehat{O}(m\epsilon^{-2})$
$n^{o(1)}$-approximate sparsest cut	$\widehat{O}(m^{1.5})$ [22]	$\tilde{O}(m)$ [18], [8]	$\widehat{O}(m)$
$n^{o(1)}$-approximate lowest-conductance cut	$\widehat{O}(m^{1.5})$ [22]	$\tilde{O}(m)$ [18], [8]	$\widehat{O}(m)$
Expander decomposition (conductance ϕ)	$\widehat{O}(m^{1.5})$ [22]	$\tilde{O}(m/\phi)$ [16] $\widehat{O}(m)$ [15], [14]	$\widehat{O}(m)$
Congestion approximator	$\Omega(m^2)$	$\tilde{O}(m)$ [25], [8], [9]	$\widehat{O}(m)$
Spectral sparsifiers	$O(mn^3\epsilon^{-2})$ [32], [33]	$\tilde{O}(m\epsilon^{-2})$ [34], [35]	$n^{o(1)}$-approximation, time $\widehat{O}(m)$
Laplacian solvers	$\tilde{O}(m^{1.31}\log(1/\epsilon))$ [36]	$\tilde{O}(m\log(1/\epsilon))$ [37]	$\widehat{O}(m\log(1/\epsilon))$

Table I

APPLICATIONS OF OUR RESULTS TO STATIC GRAPH PROBLEMS. AS USUAL, n AND m DENOTE THE NUMBER OF NODES AND EDGES OF THE INPUT GRAPH, RESPECTIVELY. WE USE $\tilde{O}$ AND $\widehat{O}$ NOTATION TO HIDE $\mathrm{polylog}n$ AND $n^{o(1)}$ FACTORS RESPECTIVELY. FOR READABILITY, WE ASSUME THAT THE WEIGHTS AND THE CAPACITIES OF EDGES ARE POLYNOMIAL IN THE PROBLEM SIZE.

Dynamic Problem	Best previous worst-case update time: deterministic	Best previous worst-case update time: randomized	Our results: deterministic
Connectivity	$O(\sqrt{n})$ [19], [20] $O(\sqrt{n}\cdot\frac{\log\log n}{\sqrt{\log n}})$ [38]	$O(\log^4 n)$ [39], [40]	$n^{o(1)}$
Minimum Spanning Forest	$O(\sqrt{n})$ [19], [20]	$n^{o(1)}$ [7]	$n^{o(1)}$

Table II

APPLICATIONS OF OUR RESULTS TO DYNAMIC (UNDIRECTED) GRAPH PROBLEMS. AS BEFORE, n AND m DENOTE THE NUMBER OF VERTICES AND EDGES OF THE INPUT GRAPH, RESPECTIVELY. FOR READABILITY, WE ASSUME THAT THE WEIGHTS AND THE CAPACITIES OF EDGES/NODES ARE POLYNOMIAL IN PROBLEM SIZE.

The first non-trivial algorithm for the Dynamic Connectivity problem dates back to Frederickson's work from 1985 [19], that provided a deterministic algorithm with $O(\sqrt{m})$ worst-case update time. Combining this algorithm with the sparsification technique of Eppstein et al. [20] yields a deterministic algorithm for Dynamic Connectivity with $O(\sqrt{n})$ worst-case update time. Improving and refining this bound has been an active research direction in the past three decades, but unfortunately, practically all follow-up results require either *randomization* or *amortization*. We now provide a summary of these results.

- **(Amortized & Randomized)** In their 1995 breakthrough paper, Henzinger and King [41] greatly improve the $O(\sqrt{n})$ worst-case update bound with a randomized Las Vegas algorithm, whose expected amortized update time is $\mathrm{poly}\log(n)$. This result has been subsequently improved, and the current best randomized algorithms have amortized update time that almost matches existing lower bounds, to within $O((\log\log n)^2)$ factors; see, e.g., [42], [43], [44], [45].
- **(Amortized & Deterministic)** Henzinger and King's 1997 deterministic algorithm [46] achieves an amortized update time of $O(n^{1/3}\log n)$. This was later substantially improved to $O(\log^2 n)$ amortized update time by the deterministic algorithm of Holm, de Lichtenberg, and Thorup [47]; this update time was in turn later improved to $O(\log^2(n)/\log\log n)$ by Wulff-Nilsen [48].
- **(Worst-Case & Randomized)** The first improvement over the $O(\sqrt{n})$ worst-case update bound was due to Kapron, King and Mountjoy [39], who provided a randomized Monte Carlo algorithm with worst-case update time $O(\log^5 n)$. This bound was later improved to $O(\log^4 n)$ by Gibb et al. [40]. Subsequently, Nanongkai, Saranurak, and Wulff-Nilsen [7], [14], [15] presented a Las Vegas algorithm for the more general dynamic MSF problem with $n^{o(1)}$ worst-case update time.

A major open problem that was raised repeatedly (see, e.g., [39], [49], [38], [50], [51], [47], [14]) is: *can we achieve an $O(n^{1/2-\epsilon})$ worst-case update time with a deterministic algorithm?* The only progress so far on this question is the deterministic algorithm of Kejlberg-Rasmussen et al. [38], that slightly improves the $O(\sqrt{n})$ worst-case update time bound to $O(\sqrt{n(\log\log n)^2/\log n})$ using word-parallelism. In this paper, we resolve this question in the affirmative, and provide a somewhat stronger result, that holds for the more general dynamic MSF problem:

Theorem 5. *There are deterministic algorithms for* Dynamic Connectivity *and* Dynamic MSF*, with $n^{o(1)}$ worst-case update time.*

In order to obtain this result, we use the algorithm of Nanongkai, Saranurak, and Wulff-Nilsen [7] for dynamic MSF. The only randomized component of their algorithm is the computation of an expander decomposition of a given graph. Since our results provide a fast deterministic algorithm for computing expander decomposition, we achieve the same $n^{o(1)}$ worst-case update time as in [7] via a deterministic algorithm.

C. Techniques

Our algorithm for the proof of Theorem 2 is based on the *cut-matching game* framework that was introduced by Khandekar, Rao and Vazirani [18], and has been used in numerous algorithms for computing sparse cuts [18], [15], [16], [22] and beyond (e.g. [52], [5], [53], [54]). Intuitively, the cut-matching game consists of two algorithms: one algorithm, called the *cut player*, needs to compute a balanced cut of a given graph that has a small value, if such a cut exists. The second algorithm, called the *matching player*, needs to solve (possibly approximately) a single-commodity maximum flow / minimum cut problem. A combination of these two algorithms is then used in order to compute a sparse cut in the input graph, or to certify that no such cut exists. Unfortunately, all current algorithms for the cut player are randomized. Our main technical contribution is an efficient deterministic algorithm that implements the cut player. The algorithm itself is recursive, and proceeds by recursively running many cut-matching games in parallel, on much smaller graphs. This requires us to adapt the algorithm of the matching player, so that it solves a somewhat harder multi-commodity flow problem. We now provide more details on the cut-matching game and on our implementation of it.

Overview of the Cut-Matching Game: We start with a high-level overview of a variant of the cut-matching game, due to Khandekar et al. [55]. We say that a graph W is a *ψ-expander* if it has no cut of sparsity less than ψ. We will informally say that W is an *expander* if it is a ψ-expander for some $\psi = 1/n^{o(1)}$. Given a graph $G = (V, E)$, the goal of the cut-matching game is to either find a balanced and sparse cut in G, or to embed an expander $W = (V, E')$ (called a *witness*) into G; note that W and G are defined over the same vertex set. The embedding of W into G needs to map every edge e of W to a path P_e in G connecting the endpoints of e. The *congestion* of this embedding is the maximum number of paths in $\{P_e \mid e \in E(W)\}$ that share a single edge of G. We require that the congestion of the resulting embedding is low. Such an embedding serves as a certificate that there is no sparse balanced cut in G. This follows from the fact that, if W is a ψ-expander, and it has a low-congestion embedding into another graph G, then G itself is a ψ'-expander, where ψ' depends on ψ and on the congestion of the embedding. The algorithm proceeds via an interaction between two algorithms, the cut player, and the matching player, and consists of $O(\log n)$ *rounds*.

At the beginning of every round, we are given a graph W whose vertex set is V, and its embedding into G; at the beginning of the first round, W contains the set V of vertices and no edges. In every round, the cut player either:

(C1) "cuts W", by finding a balanced sparse cut S in W; or
(C2) "certifies W" by announcing that W is an expander.

If W is certified (Item (C2)), then we have constructed the desired embedding of an expander into G, so we can terminate the algorithm and certify that G has no balanced sparse cut. If a cut S is found in W (Item (C1)), then we invoke the matching player, who either:

(M1) "matches W", by adding to W a large matching $M \subseteq S \times (V \setminus S)$ that can be embedded into G with low congestion; or
(M2) "cuts G", by finding a balanced sparse cut T in G (the cut T is intuitively what prevents the matching player from embedding a large matching $M \subseteq S \times (V \setminus S)$ into G).

If a sparse balanced cut T is found in graph G (Item (M2)), then we return this cut and terminate the algorithm. Otherwise, the game continues to the next round. It was shown in [55] that the algorithm must terminate after $\Theta(\log n)$ rounds.

In the original cut-matching game by Khandekar, Rao and Vazirani [18], the matching player was implemented by an algorithm that computes a single-commodity maximum flow / minimum cut. The algorithm for the cut player was defined somewhat differently, in that in the case of Item (C1), the cut that it produced was not necessarily sparse, but it still had some useful properties, which guaranteed that the algorithm terminates after $O(\log^2 n)$ iterations. In order to implement the cut player, the algorithm of [18] (implicitly) considers n vectors of dimension n each, that represent the probability distributions of random walks on the witness graph, starting from different vertices of G, and then uses a random projection of these vectors in order to construct the balanced cut. The algorithm exploits the properties of the witness graph in order to compute these projections efficiently, without explicitly constructing these vectors, which would be too time consuming. Previous work (see, e.g., [16], [17]) implies that one can use algorithms for computing *maximal* flows instead of maximum flows in order to implement the matching player in near-linear time deterministically, if the target parameters $1/\phi, \alpha \leq n^{o(1)}$. This still left open the question: *can the cut player be implemented via a deterministic and efficient algorithm?*

A natural strategy for derandomizing the algorithm of [18] for the cut player is to avoid the random projection of the vectors. In a previous work of a subset of the authors

with Yingchareonthawornchai [22], this idea was used to develop a fast PageRank-based algorithm for the cut player, that can be viewed as a derandomization of the algorithm of Andersen, Chung and Lang for balanced sparse cut [21]. Unfortunately, it appears that this technique cannot lead to an algorithm whose running time is below $\Theta(n^2)$: if we cannot use random projections, then we need to deal with n vectors of dimension n each when implementing the cut player, and so the running time of $\Omega(n^2)$ seems inevitable. In this paper, we implement the cut player in a completely different way from the previously used approaches, by solving the balanced sparse cut problem recursively.

We start by observing that, in order to implement the cut player via the approach of [55], it is sufficient to provide an algorithm for computing a balanced sparse cut on the witness graph W; in fact, it is not hard to see that it is sufficient to solve this problem approximately. However, this leads us to a chicken-and-egg situation, where, in order to solve the Minimum Balanced Cut problem on the input graph G, we need to solve the Minimum Balanced Cut problem on the witness graph W. While graph W is guaranteed to be quite sparse (with maximum vertex degree $O(\log n)$), it is not clear that solving the Minimum Balanced Cut problem on this graph is much easier.

This motivates our recursive approach, in which, in order to solve the Minimum Balanced Cut problem on the witness graph W, we run a large number of cut-matching games in it simultaneously, each of which has a separate witness graph, containing significantly fewer vertices. It is then sufficient to solve the Minimum Balanced Cut problem on each of the resulting, much smaller, witness graphs. We prove the following theorem that provides a deterministic algorithm for the cut player via this recursive approach.

Theorem 6. *There is an universal constant N_0, and a deterministic algorithm, that we call* CUTORCERTIFY*, that, given an n-vertex graph $G=(V,E)$ with maximum vertex degree $O(\log n)$, and a parameter $r \geq 1$, such that $n^{1/r} \geq N_0$, returns one of the following:*

- *either a cut (A,B) in G with $|A|,|B| \geq n/4$ and $|E_G(A,B)| \leq n/100$; or*
- *a subset $S \subseteq V$ of at least $n/2$ vertices, such that $\Psi(G[S]) \geq 1/\log^{O(r)} n$.*

The running time of the algorithm is $O\left(n^{1+O(1/r)} \cdot (\log n)^{O(r^2)}\right)$.

We note that a somewhat similar recursive approach was used before, e.g., in Madry's construction of j-trees [56], and in the recursive construction of short cycle decompositions [57], [58]. In fact, [22] use Madry's j-trees to solve Minimum Balanced Cut by running cut-matching games on graphs containing fewer and fewer nodes, obtaining an $(m^{1.5+o(1)})$-time algorithm. Unfortunately, improving this bound further does not seem viable via this approach, since the total number of edges contained in the graphs that belong to deeper recursive levels is very large. Specifically, assume that we are given an n-node graph G with m edges, together with a parameter $k \geq 1$. We can then use the j-trees in order to reduce the problem of computing Minimum Balanced Cut on G to the problem of computing Minimum Balanced Cut on k graphs, each of which contains roughly n/k nodes. Unfortunately, each of these graphs may have $\Omega(m)$ edges. Therefore, the total number of edges in all resulting graphs may be as large as $\Omega(mk)$, which is one of the major obstacles to obtaining faster algorithms for Minimum Balanced Cut using j-trees.

We now provide a more detailed description of the new recursive strategy that we use in order to prove Theorem 6.

New Recursive Strategy: We partition the vertices of the input n-vertex graph G into k subsets $V_1, V_2, \ldots, V_k$ of roughly equal cardinality, for a large enough parameter k (for example, $k = n^{o(1)}$). The algorithm consists of two stages. In the first stage, we attempt to construct k expander graphs $W_1, \ldots, W_k$, where $V(W_i) = V_i$ for all $1 \leq i \leq k$, and embed them into the graph G simultaneously. If we fail to do so, then we will compute a sparse balanced cut in G. In order to do so, we run k cut-matching games in parallel. Specifically, we start with every graph W_i containing the set V_i of vertices and no edges, and then perform $O(\log n)$ iterations. In every iteration, we run the CUTORCERTIFY algorithm on each graph $W_1, \ldots, W_k$ in parallel. Assume that for all $1 \leq i \leq k$, the algorithm returns a sparse balanced cut (A_i, B_i) in W_i. We then use an algorithm of the matching player, that either computes, for each $1 \leq i \leq k$, a matching M_i between vertices of A_i and B_i, and computes a low-congestion embedding of all matchings $M_1, \ldots, M_k$ into graph G simultaneously, or it returns a sparse balanced cut in G. In the former case, we augment each graph W_i by adding the set M_i of edges to it. In the latter case, we terminate the algorithm and return the sparse balanced cut in graph G as the algorithm's output. If the algorithm never terminates with a sparse balanced cut, then we are guaranteed that, after $O(\log n)$ iterations, the graphs $W_1, \ldots, W_k$ are all expanders (more precisely, each of these graphs contains a large enough expander, but we ignore this technicality in this informal overview), and moreover, we obtain a low-congestion embedding of the disjoint union of these graphs into G. Note that, in order to execute this stage, we recursively apply algorithm CUTORCERTIFY to k graphs, whose sizes are significantly smaller than the size of the graph G.

In the second stage, we attempt to construct a single expander graph W^* on the set $\{v_1, \ldots, v_k\}$ of vertices, where for each $1 \leq i \leq k$, we view vertex v_i as representing the set V_i of vertices of G. We also attempt to embed the graph W^* into G, where every edge $e = (v_i, v_j)$ is embedded into $\Omega(n/k)$ paths connecting vertices of V_i to vertices of V_j. In

order to do so, we start with the graph W^* containing the set $\{v_1, \ldots, v_k\}$ of vertices and no edges and then iterate. In every iteration, we run algorithm CUTORCERTIFY on the current graph W^*, obtaining a partition (A, B) of its vertices. We then use an algorithm of the matching player in order to compute a matching M between vertices of A and vertices of B, and to embed every edge $(v_i, v_j) \in M$ of the matching into $\Omega(n/k)$ paths connecting vertices of V_i to vertices of V_j in graph G, with low congestion. If we do not succeed in computing the matching and the embedding, then the algorithm of the matching player returns a sparse balanced cut in graph G. We then terminate the algorithm and return this cut as the algorithm's output. Otherwise, we add the edges of M to graph W^* and continue to the next iteration. The algorithm terminates once graph W^* is an expander, which must happen after $O(\log n)$ iterations.

Lastly, we compose the expanders $W_1, \ldots, W_k$ and W^* in order to obtain an expander graph $\hat{W}$ that embeds into G with low congestion; the embedding is obtained by combining the embeddings of the graphs $W_1, \ldots, W_k$ and the embedding of graph W^*. This serves as a certificate that G is an expander graph.

Note that the algorithm for the matching player that we need to use differs from the standard one in that it needs to compute k different matchings between k different pre-specified pairs of vertex subsets. Specifically, the algorithm for the matching player is given k pairs $(A_1, B_1), \ldots, (A_k, B_k)$ of subsets of vertices of G of equal cardinality. Ideally, we would like the algorithm to either (i) compute, for all $1 \leq i \leq k$, a perfect matching M_i between vertices of A_i and vertices of B_i, and embed all edges of $M_1 \cup \cdots \cup M_k$ into G simultaneously with low congestion; or (ii) compute a sparse balanced cut in G. In fact our algorithm for the matching player achieves a somewhat weaker objective: namely, the matchings M_i are not necessarily perfect matchings, but they are sufficiently large. In order to overcome this difficulty, we introduce "fake" edges that augment each matching M_i to a perfect matching. As a result, if the algorithm fails to compute a sparse balanced cut in G, then we are only guaranteed that $G \cup F$ is an expander, where F is (a relatively small) set of fake edges. We then use a known "expander trimming" algorithm of [16] in order to find a large subset $S \subseteq V(G)$ of vertices, such that $G[S]$ is an expander, and the cut S is sufficiently sparse. We note that the notion of fake edges was used before in the context of the cut-matching game, e.g. in [18].

The algorithm of the matching player builds on the idea of Chuzhoy and Khanna [17] of computing maximal sets of short edge-disjoint paths, which can be implemented efficiently via Even-Shiloach's algorithm for decremental single-source shortest paths [59]. Unfortunately, this approach results in a somewhat slower running time of $O\left(m^{1+O(1/r)} \cdot (\log m)^{O(r^2)}/\phi^2\right)$, introducing a quadratic dependence on $1/\phi$, where ϕ is the conductance parameter. The expander trimming algorithm of [16] that is exploited by the cut player also unfortunately introduces a linear dependence on $1/\phi$ in the running time. As a result, we obtain an algorithm for the BalCutPrune problem that is sufficiently fast in the high-conductance regime, that is, where $\phi = 1/\mathrm{poly}\log n$, but is too slow for the setting where the parameter ϕ is low. Luckily, the high-conductance regime is sufficient for many of our applications, and in particular it allows us to obtain efficient approximation algorithms for maximum flow. This algorithm can then in turn be used in order to implement the matching player, even in the low-conductance regime, removing the dependence of the algorithm's running time on ϕ. Additional difficulty for the low-conductance regime is that we can no longer afford to use the expander trimming algorithm of [16]. Instead, we provide an efficient deterministic bi-criteria approximation algorithm for the most-balanced sparsest cut problem, and use this algorithm in order to solve the BalCutPrune problem in the low-conductance regime. This part closely follows ideas of [15], [14], [17], [60].

Due to lack of space, all formal proofs and further details are deferred to the full version of the paper, that is available at https://arxiv.org/abs/1910.08025.

II. Open Problems

A very interesting remaining open problem is to obtain deterministic algorithms for Minimum Balanced Cut, Sparsest Cut and Lowest-Conductance Cut, that achieve a polylogarithmic approximation ratio, with running time $O(m^{1+o(1)})$. It would also be interesting to obtain deterministic $n^{o(1)}$-approximation algorithms for these problems with running time $\tilde{O}(m)$. The latter result would imply a near-linear time deterministic algorithm for computing an expander decompositions, matching the performance of the best current randomized algorithm of [16]).

It is typically desirable for a dynamic graph algorithm to have worst-case update time $O(\mathrm{poly}\log n)$, where n is the number of vertices of the input graph. Our result for dynamic connectivity (Theorem 5) only guarantees $n^{o(1)}$ worst-case update time, leaving the question of designing a deterministic algorithm with $O(\mathrm{poly}\log n)$ worst-case update time for dynamic connectivity as a major open problem. In fact, it would even be interesting to achieve such bounds with a Las Vegas randomized algorithm. It would also be interesting to design a Monte Carlo randomized algorithm for maintaining a spanning forest with $O(\mathrm{poly}\log n)$ worst-case update time, that does rely on the so-called *oblivious adversary assumption*.[4] We remark that even if one can implement an algorithm for Theorem 2 with running time

[4]It was shown by Kapron et al. [39], that a spanning forest can be maintained in $O(\mathrm{poly}\log n)$ worst case update time by a Monte Carlo randomized algorithm under the oblivious adversary assumption.

$\tilde{O}(m)$ and approximation factor $O(\text{polylog}n)$, this would not immediately imply any of the above goals. The reason is that there are several additional components in the algorithm of Nanongkai et al. [7] that each incur the $n^{o(1)}$ factor in the update time.

Our deterministic algorithm for spectral sparsifiers only achieves a factor $n^{o(1)}$-approximation. It is an intriguing open question whether $(1+\epsilon)$-approximate cut/spectral sparsifiers can be computed deterministically in almost-linear time. It is also interesting whether there is a deterministic $O(\sqrt{\log n})$-approximation algorithm for Lowest-Conductance Cut, whose running time matches that of the best currently known randomized algorithm, which is $O(m^{1+\epsilon})$, for any constant $\epsilon > 0$ [61]. We believe that resolving both questions would require significantly new ideas.

Acknowledgements

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 was also partially supported by the Swedish Research Council (Reg. No. 2015-04659). Chuzhoy was supported in part by NSF grant CCF-1616584. Gao and Peng were supported in part by NSF grant CCF-1718533. Li was supported in part by NSF award CCF-1907820.

References

[1] H. Räcke, "Minimizing congestion in general networks," in *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, 2002, pp. 43–52.

[2] D. A. Spielman and S. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *STOC*. ACM, 2004, pp. 81–90.

[3] L. Trevisan, "Approximation algorithms for unique games," in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*. IEEE, 2005, pp. 197–205.

[4] S. Arora, E. Hazan, and S. Kale, "$O(\sqrt{\log n})$ approximation to SPARSEST CUT in $\tilde{O}(n^2)$ time," *SIAM J. Comput.*, vol. 39, no. 5, pp. 1748–1771, 2010. [Online]. Available: https://doi.org/10.1137/080731049

[5] H. Räcke, C. Shah, and H. Täubig, "Computing cut-based hierarchical decompositions in almost linear time," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, 2014, pp. 227–238. [Online]. Available: https://doi.org/10.1137/1.9781611973402.17

[6] K. Kawarabayashi and M. Thorup, "Deterministic edge connectivity in near-linear time," *J. ACM*, vol. 66, no. 1, pp. 4:1–4:50, 2019. [Online]. Available: https://doi.org/10.1145/3274663

[7] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen, "Dynamic minimum spanning forest with subpolynomial worst-case update time," in *FOCS*. IEEE Computer Society, 2017, pp. 950–961.

[8] J. Sherman, "Nearly maximum flows in nearly linear time," in *FOCS*. IEEE Computer Society, 2013, pp. 263–269.

[9] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford, "An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, 2014, pp. 217–226.

[10] R. Peng, "Approximate undirected maximum flows in $O(m\text{poly}\log(n))$ time," in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, 2016, pp. 1862–1867. [Online]. Available: https://doi.org/10.1137/1.9781611974331.ch130

[11] R. Kyng, R. Peng, S. Sachdeva, and D. Wang, "Flows in almost linear time via adaptive preconditioning," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, 2019, pp. 902–913. [Online]. Available: https://doi.org/10.1145/3313276.3316410

[12] M. B. Cohen, A. Madry, P. Sankowski, and A. Vladu, "Negative-weight shortest paths and unit capacity minimum cost flow in $O(m^{10/7}\log W)$ time (extended abstract)," in *SODA*. SIAM, 2017, pp. 752–771.

[13] A. Madry, "Computing maximum flow with augmenting electrical flows," in *FOCS*. IEEE Computer Society, 2016, pp. 593–602.

[14] 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: https://doi.org/10.1145/3055399.3055415

[15] 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: https://doi.org/10.1145/3055399.3055447

[16] T. Saranurak and D. Wang, "Expander decomposition and pruning: Faster, stronger, and simpler," in *SODA*. SIAM, 2019, pp. 2616–2635.

[17] J. Chuzhoy and S. Khanna, "A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems," in *STOC*. ACM, 2019, pp. 389–400.

[18] R. Khandekar, S. Rao, and U. V. Vazirani, "Graph partitioning using single commodity flows," *J. ACM*, vol. 56, no. 4, pp. 19:1–19:15, 2009. [Online]. Available: https://doi.org/10.1145/1538902.1538903

[19] 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

[20] 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.

[21] R. Andersen, F. R. K. Chung, and K. J. Lang, "Using pagerank to locally partition a graph," *Internet Mathematics*, vol. 4, no. 1, pp. 35–64, 2007. [Online]. Available: https://doi.org/10.1080/15427951.2007.10129139

[22] Y. Gao, J. Li, D. Nanongkai, R. Peng, T. Saranurak, and S. Yingchareonthawornchai, "Deterministic graph cuts in subquadratic time: Sparse, balanced, and k-vertex," *arXiv preprint arXiv:1910.07950*, 2019.

[23] L. Orecchia and N. K. Vishnoi, "Towards an sdp-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, 2011, pp. 532–545. [Online]. Available: https://doi.org/10.1137/1.9781611973082.42

[24] L. Orecchia, S. Sachdeva, and N. K. Vishnoi, "Approximating the exponential, the lanczos method and an $\tilde{o}(m)$-time spectral algorithm for balanced separator," in *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, 2012, pp. 1141–1160. [Online]. Available: https://doi.org/10.1145/2213977.2214080

[25] A. Madry, "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms," in *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, 2010, pp. 121–130. [Online]. Available: https://doi.org/10.1145/1806689.1806708

[26] S. Arora, S. Rao, and U. V. Vazirani, "Expander flows, geometric embeddings and graph partitioning," *J. ACM*, vol. 56, no. 2, pp. 5:1–5:37, 2009. [Online]. Available: https://doi.org/10.1145/1502793.1502794

[27] L. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIAM J. Discrete Math.*, vol. 13, no. 4, pp. 505–520, 2000, announced at FOCS'99. [Online]. Available: https://doi.org/10.1137/S0895480199355754

[28] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," *ACM Trans. Algorithms*, vol. 4, no. 1, pp. 13:1–13:17, 2008. [Online]. Available: https://doi.org/10.1145/1328911.1328924

[29] N. Alon, "Eigenvalues and expanders," *Combinatorica*, vol. 6, no. 2, pp. 83–96, 1986. [Online]. Available: https://doi.org/10.1007/BF02579166

[30] A. V. Goldberg and S. Rao, "Beyond the flow decomposition barrier," *J. ACM*, vol. 45, no. 5, pp. 783–797, 1998. [Online]. Available: https://doi.org/10.1145/290179.290181

[31] J. Sherman, "Area-convexity, l_∞ regularization, and undirected multicommodity flow," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, 2017, pp. 452–460. [Online]. Available: https://doi.org/10.1145/3055399.3055501

[32] J. Batson, D. A. Spielman, and N. Srivastava, "Twice-Ramanujan sparsifiers," *SIAM Journal on Computing*, vol. 41, no. 6, pp. 1704–1721, 2012.

[33] M. K. de Carli Silva, N. J. A. Harvey, and C. M. Sato, "Sparse sums of positive semidefinite matrices," *ACM Trans. Algorithms*, vol. 12, no. 1, pp. 9:1–9:17, 2016. [Online]. Available: https://doi.org/10.1145/2746241

[34] D. A. Spielman and S. Teng, "Spectral sparsification of graphs," *SIAM J. Comput.*, vol. 40, no. 4, pp. 981–1025, 2011. [Online]. Available: https://doi.org/10.1137/08074489X

[35] Y. T. Lee and H. Sun, "An sdp-based algorithm for linear-sized spectral sparsification," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, 2017, pp. 678–687. [Online]. Available: https://doi.org/10.1145/3055399.3055477

[36] D. A. Spielman and S. Teng, "Solving sparse, symmetric, diagonally-dominant linear systems in time $0(m^{1.31})$," in *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, 2003, pp. 416–427. [Online]. Available: https://doi.org/10.1109/SFCS.2003.1238215

[37] ——, "Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," *SIAM J. Matrix Analysis Applications*, vol. 35, no. 3, pp. 835–885, 2014.

[38] 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*, 2016, pp. 53:1–53:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.ESA.2016.53

[39] 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*, 2013, pp. 1131–1142. [Online]. Available: https://doi.org/10.1137/1.9781611973105.81

[40] 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

[41] 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

[42] 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.

[43] 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

[44] 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

[45] 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

[46] M. R. Henzinger and V. King, "Maintaining minimum spanning trees in dynamic graphs," in *ICALP*, ser. Lecture Notes in Computer Science, vol. 1256. Springer, 1997, pp. 594–604.

[47] 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

[48] 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

[49] M. Patrascu and M. Thorup, "Planning for fast connectivity updates," in *FOCS*. IEEE Computer Society, 2007, pp. 263–271.

[50] V. King, "Fully dynamic connectivity," in *Encyclopedia of Algorithms*, 2016, pp. 792–793.

[51] ——, "Fully dynamic connectivity," in *Encyclopedia of Algorithms*. Springer, 2008.

[52] C. Chekuri and J. Chuzhoy, "Large-treewidth graph decompositions and applications," in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, 2013, pp. 291–300. [Online]. Available: https://doi.org/10.1145/2488608.2488645

[53] ——, "Polynomial bounds for the grid-minor theorem," *J. ACM*, vol. 63, no. 5, pp. 40:1–40:65, 2016. [Online]. Available: https://doi.org/10.1145/2820609

[54] J. Chuzhoy and S. Li, "A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2," *J. ACM*, vol. 63, no. 5, pp. 45:1–45:51, 2016. [Online]. Available: https://doi.org/10.1145/2893472

[55] R. Khandekar, S. Khot, L. Orecchia, and N. K. Vishnoi, "On a cut-matching game for the sparsest cut problem," *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 2007.

[56] A. Madry, "Fast approximation algorithms for cut-based problems in undirected graphs," in *FOCS*. IEEE Computer Society, 2010, pp. 245–254.

[57] T. Chu, Y. Gao, R. Peng, S. Sachdeva, S. Sawlani, and J. Wang, "Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions," in *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, 2018, pp. 361–372. [Online]. Available: https://doi.org/10.1109/FOCS.2018.00042

[58] Y. P. Liu, S. Sachdeva, and Z. Yu, "Short cycles via low-diameter decompositions," in *SODA*. SIAM, 2019, pp. 2602–2615.

[59] S. Even and Y. Shiloach, "An on-line edge-deletion problem," *Journal of the ACM (JACM)*, vol. 28, no. 1, pp. 1–4, 1981.

[60] Y. Chang and T. Saranurak, "Improved distributed expander decomposition and nearly optimal triangle enumeration," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019.*, 2019, pp. 66–73. [Online]. Available: https://doi.org/10.1145/3293611.3331618

[61] J. Sherman, "Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$-approximations to sparsest cut," in *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, 2009, pp. 363–372.